②

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS
N/A

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)
AFOSR · TR · 89 - 0841

6a. NAME OF PERFORMING ORGANIZATION
QUALCOMM, Inc.

6b. OFFICE SYMBOL (If applicable)
IDN14

7a. NAME OF MONITORING ORGANIZATION
AFOSR, Directorate of Mathematical and Information Sciences

6c. ADDRESS (City, State and ZIP Code)
10555 Sorrento Valley Road
San Diego, CA 92121

7b. ADDRESS (City, State and ZIP Code)
Building 410
Bolling AFB, DC 20332-6448

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
USAF, AFSC

8b. OFFICE SYMBOL (If applicable)

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F49620-88-C-0088

8c. ADDRESS (City, State and ZIP Code)
Air Force Office of Scientific Research
Building 410
Bolling AFB, DC 20332-6448

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2305	A1	

11. TITLE (Include Security Classification)
Research in Mathematics & Computer Science

12. PERSONAL AUTHOR(S)
Wolf, Jack Keil - Fredrickson, Lyle J. - Viterbi, Andrew James

13a. TYPE OF REPORT
Final Technical Report

13b. TIME COVERED
FROM 88AUG01 TO 89JAN31

14. DATE OF REPORT (Yr., Mo., Day)
89 MAR 31

15. PAGE COUNT
54

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Cyclic redundancy check codes (or CRC codes) have become the standard means for insuring the integrity of messages that have been transmitted over a noisy communications channel. The sole purpose of these codes is to detect transmission errors (in contrast to error correction codes (or ECC codes) which attempt to correct transmissions in errors). Sometimes both CRC and ECC codes are utilized and in that case the burden is on the CRC code to detect errors that were not correctly decoded by the ECC code.

Unfortunately, even the very best CRC codes cannot detect all transmission errors. The probability of CRC failure is called the probability of undetected error. The thrust of this study was concerned with finding an efficient method of calculating this probability of undetected error and then to use this method to find good (or even the best) CRC codes.

(continued on sheet 2)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Viterbi, Andrew James

22b. TELEPHONE NUMBER (Include Area Code)
(619) 587-1121

22c. OFFICE SYMBOL
N/A

EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

(continued from sheet 1 of DD Form 1473, 83 APR)

19. A new algorithm was implemented to find good choices for the generator polynomial of CRC codes, that is, generator polynomials for which the probability of undetected error was less than a given bound for all shortened block lengths and for all values of the binary symmetric channel error rate.

Results are given for generator polynomials corresponding to 8, 16, 24 and 32 parity bits. All possible generator polynomial corresponding to 8 and 16 parity bits and some of the generator polynomials corresponding to 24 and 32 parity bits were tested.

TABLE OF CONTENTS

I. INTRODUCTION 1

II. CALCULATING THE PROBABILITY OF UNDETECTED ERROR FOR SHORTENED CRC CODES 4

III. PROGRAM FOR DETERMINING WEIGHT DISTRIBUTION AND ELIMINATING IMPROPER CRCS 10

III.1 (LISTING) C PROGRAM FOR ELIMINATING IMPROPER POLYNOMIALS 12

IV. ASSEMBLER CODE FOR CRITICAL SUBROUTINE 15

IV.1 (LISTING) 68000 ASSEMBLER SUBROUTINE FOR DETERMINING WEIGHT ENUMERATIONS 16

V. SAMPLING THE WEIGHT DISTRIBUTION 19

VI. HARDWARE TESTER FOR CRC CODES 30

VII. EVALUATION OF 8 BIT CRCS 33

VIII. EVALUATION OF 16 BIT CRCS 34

IX. EVALUATION OF 24 BIT CRCS 38

X. EVALUATION OF 32 BIT CRCS 44

XI. CONCLUSION 48

XII. REFERENCES 49

XIII. PROFESSIONAL PERSONNEL 50

LIST OF FIGURES

FIGURE 1. LINEAR FEEDBACK SHIFT REGISTER (LFSR) FOR OPERATION MODULO P(X). 9

FIGURE 2. CODE WORD VIEWER FOR DUAL OF CODE C_2. 9

FIGURE 3. CODE WEIGHT VIEWER FOR DUAL OF CODE C_2. 9

FIGURE 4. INITIALIZATION LOOP OF ASSEMBLER SUBROUTINE FLOWCHART 17

FIGURE 5. MAIN LOOP OF ASSEMBLER SUBROUTINE FLOWCHART 18

FIGURE 6. CONTEMPLATED HARDWARE 31

FIGURE 7. HARDWARE MODES OF OPERATION 32

FIGURE 8. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 16 BIT CRCS AT BLOCK LENGTH N = 256 35

FIGURE 9. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 16 BIT CRCS AT BLOCK LENGTH N = 1024 36

FIGURE 10. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 16 BIT CRCS AT BLOCK LENGTH N = 4096 37

FIGURE 11. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 24 BIT CRCS AT BLOCK LENGTH N = 32 39

FIGURE 12. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 24 BIT CRCS AT BLOCK LENGTH N = 256 40

FIGURE 13. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 24 BIT CRCS AT BLOCK LENGTH N = 1024 41

FIGURE 14. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 24 BIT CRCS AT BLOCK LENGTH N = 4096 42

FIGURE 15. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR CRC-24Q AT VARIOUS BLOCK LENGTHS 43

FIGURE 16. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 32 BIT CRCS AT BLOCK LENGTH N = 1024 45

FIGURE 17. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 32 BIT CRCS AT BLOCK LENGTH N = 256 46

FIGURE 18. PROBABILITY OF UNDETECTED BLOCK ERROR VS. PROBABILITY OF BIT ERROR FOR 32 BIT CRCS AT BLOCK LENGTH N = 4096 47

List of Tables

TABLE 1. SAMPLED WEIGHT DISTRIBUTIONS FOR 24 BIT CRCS AT BLOCK LENGTH N = 32 21

TABLE 2. SAMPLED WEIGHT DISTRIBUTIONS FOR 24 BIT CRCS AT BLOCK LENGTH N = 1024 24

TABLE 3. VARIANCE OF SAMPLE WEIGHT DISTRIBUTIONS FOR 24 BIT CRCS AT BLOCK LENGTH N = 32 29

AFOSR-TR- 89-0841

QUALCOMM, Inc.
10555 Sorrento Valley Road
San Diego, California 92121

FINAL TECHNICAL REPORT FOR THE RESEARCH IN MATHEMATICS AND COMPUTER SCIENCE: CALCULATION OF THE PROBABILITY OF UNDETECTED ERROR FOR CERTAIN ERROR DETECTION CODES

PHASE I

31 March 1989

Submitted To:

USAF, AFSC
Air Force Office of Scientific Research
Building 410
Bolling Air Force Base, DC 20332-6448

Contract No. F49620-88-C-0088

I. INTRODUCTION

Cyclic redundancy check codes (or CRC codes) have become the standard means for insuring the integrity of messages that have been transmitted over a noisy communications channel. The sole purpose of these codes is to detect transmission errors (in contrast to error correction codes (or ECC codes) which attempt to correct transmissions in errors). Sometimes both CRC and ECC codes are utilized and in that case the burden is on the CRC code to detect errors that were not correctly decoded by the ECC code.

Unfortunately even the very best CRC codes cannot detect all transmission errors. In particular, all that we as communication engineers can insist upon is that the probability that a CRC code fails in its attempt to detect transmission errors is less than some prescribed value. The probability of CRC failure is called the probability of undetected error and is denoted by the symbol P_{ud}. The thrust of this study was concerned with finding an efficient method of calculating this probability of undetected error and then to use this method to find good (or even the best) CRC codes.

It should be emphasized that the value of the probability of undetected error depends upon the statistical model one assume for the errors. Here we have assumed a random error model where the probability of error on every binary digit is ε ($o \leq \varepsilon \leq 1/2$) and the errors on different binary digits are statistically independent of one another. This model, called a binary symmetric channel (or BSC) was assumed since

(a) it is a model that describes some commonly used modulation/channel/detection schemes, such as BPSK or QPSK modulation in an additive white Gaussian noise channel with an optimum receiver,

(b) correlated errors can always be converted to independent errors by interleaving at a sufficient depth, and

(c) channel models for correlated errors describe some specific type of correlation and are not general enough to describe the general class of non-random errors.

It should be realized that there are many situations where CRC codes are expected to detect errors which are not random in nature. One such example is when a CRC code is used to detect errors that were miscorrected by an ECC code. (The ECC decoder will produce burst errors). As stated above, one manner for treating this problem is to interleave to a sufficient depth so that the errors within a CRC block are statistically independent. If interleaving is not used, however, the results of this study cannot be directly applied.

Although any code can be utilized as an error detection code, a particular class of cyclic (or shortened cyclic) codes called CRC codes, have become the de facto (or true) standards. These codes are binary codes with generator polynomials of the form $g(x) = (x+1)\,p(x)$ where $p(x)$ is a primitive irreducible polynomial of degree $(r - 1)$. Thus $g(x)$ has degree r where r is the number of parity bits per block.

Three of these codes have become international standards [1].
These codes are

CRC-12	$p(x) = x^{11} + x^2 + 1$
CRC-16	$p(x) = x^{15} + x + 1$
CRC-CCITT	$p(x) = x^{15} + x^{14} + x^{13} + x^{12} + x^4 + x^3 + x^2 + x + 1$

The natural block length of a CRC code with r parity bits and generator polynomial

$$g(x) = (x+1)\ p(x)$$

is $2^{r-1} - 1$. Thus the CRC-12 code has natural block length 2047 and the CRC-16 and CRC-CCITT codes have natural block length 32,767.

Often the natural block length is longer than the desired block length of the code. This is not a problem since any code can be "shortened" to any block length smaller than the natural block length. If one desires to shorten a code by "x" digits one artificially sets (the first) "x" information digits to zero and then omits these "x" digits in transmission or storage. These "x" digits can then be inserted without error at the receiver. Certainly the error correction capability of the shortened code is as least as good as that of the original unshortened code. In particularly, shortening a code in the manner described above cannot decrease the minimum distance of the code and even can increase this minimum distance. Surprisingly, the same statement cannot be made with regard to the probability of undetected error for the code. In particular, it has been found that for a fixed channel bit error probability, the probability of undetected error for a shortened CRC code can be worse than (that is, greater than) the probability of undetected error for the unshortened code.

This surprising result is the motivation for this study. Let $P_{ud}(n,p)$ be the probability of undetected error for a CRC code with (shortened or unshortened) block length n and channel (random) bit error probability p. It is easy to show that the probability of undetected error for an unshortened CRC code (with generator polynomial $g(x) = (x+1)\ p(x)$, degree $[g(x)] = r$, and natural block length $n = 2^{r-1} - 1$) is given as

$$P_{ud}\ (2^{r-1} - 1, p) = 2^{-r} \sum_{i=0}^{n} B_i\ (1 - 2p)^i - (1 - p)^n.$$

where

$$B_i = \begin{cases} 1 & i = 0;\ i = 2^{r-1} - 1 \\ 2^{r-1} - 1 & i = 2^{r-2} - 1;\ \ i = 2^{r-2} \\ 0 & \text{otherwise} \end{cases}$$

and $n = 2^{r-1} - 1$.

It is easily verified that

$$P_{ud}\ (2^{r-1} - 1,\ p) < 2^{-r}$$

for all values of the channel bit error probability p in the range $0 \le p \le 1/2$. It should be noted that this result holds for <u>all</u> CRC codes at their natural block length irrespective of the choice of the primitive polynomial p(x). In particular, we find that for some choices of p(x), $P_{ud}(n,p)$ will be less than 2^{-r} for all shortened block lengths and for all p in the range $0 \le p \le 1/2$ while for other choices of p(x), $P_{ud}(n,p)$ can be much greater than 2^{-r} for some choices of the shortened block length and for some values of the channel bit error probability p.

One purpose of this study was to find good choices for the generator polynomial of CRC codes, that is, generator polynomials for which $P_{ud}(n,p)$ was less than 2^{-r} for all shortened block lengths and for all values of p. In the next section we describe and give a mathematical justification for an algorithm for finding good CRC polynomials. We also describe fast versions of this algorithm and give the computer code for the "speed-up" versions of this algorithm. We then give the results of applying this algorithm to generator polynomials corresponding to 8, 16, 24 and 32 parity bits. All possible generator polynomial corresponding to 8 and 16 parity bits and some of the generator polynomials corresponding to 24 and 32 parity bits were tested. Also results for an incomplete running of the algorithm were compared with results for a full running of the algorithm. Also, a hardware tester for polynomial of higher degree is described. The report ends with a set of conclusions.

II. CALCULATING THE PROBABILITY OF UNDETECTED ERROR FOR SHORTENED CRC CODES

An unshortened cyclic code C of block length n may be defined in terms of a *generator polynomial* g(x), or equivalently in terms of its *parity-check polynomial*, h(x). The two polynomials are related by the equation

$$g(x)h(x) = x^n - 1. \tag{1}$$

For binary cyclic codes, the coefficients of the polynomials are either 0 or 1, and coefficient addition is performed modulo 2. We can therefore equate coefficient subtraction and addition, and replace (1) with the equivalent relation

$$g(x)h(x) = x^n + 1. \tag{2}$$

Let the degree of the generator polynomial, deg[g(x)] = r. Let k = n - r.

We can therefore give two equivalent definitions for vectors which are in the code. A vector of n bits, $\{c_0, c_1, \dots, c_{n-1}\}$, is a *code vector* if and only if the *code polynomial*

$$c(x) = \sum_{i=0}^{n-1} c_i x^i$$

is the generator polynomial multiplied by a polynomial of degree (k - 1) or less [2]. That is,

$$c(x) = g(x)\,a(x) \text{ where } \deg[a(x)] \leq k - 1. \tag{3}$$

Equivalently, c(x) is a code polynomial if and only if

$$c(x)\,h(x) \equiv 0 \quad \text{modulo}(x^n + 1) \tag{4}$$

since

$$c(x)\,h(x) = a(x)\,g(x)\,h(x) = a(x)(x^n + 1) \equiv 0 \quad \text{modulo}(x^n + 1).$$

Since there are a total of k arbitrary coefficients in a(x), the code takes k arbitrary bits of information and appends r = n - k bits of redundancy, is referred to as an (n,k) code, and has 2^k code words.

The *dual code*, *D*, of the cyclic code, *C*, is defined by swapping the roles of g(x) and h(x). That is, the dual code, *D*, has generator polynomial h(x). Since the degree of h(x),

$$\deg[h(x)] = k, \tag{5}$$

and the block length remains unchanged, the dual code, *D*, is an (n,r) code, and has 2^r code words.

For any (n,m) binary linear code *C*, the probability of undetected block error, P_{ud}, is given in terms of the random bit error probability, p, by

$$P_{ud} = \sum_{i=1}^{n} A_i\, p^i(1-p)^{n-i} \qquad (6)$$

where A_i is the number of code words with exactly i ones in the code C [3]. The numbers $\{A_i\}$ are referred to as the weight enumerators of the code C. Let B_i be the number of code words with exactly i ones in the dual code, D, of the code C.

By use of the MacWilliam's Identities [4], we can express P_{ud} in terms of the $\{B_i\}$. That is,

$$P_{ud} = 2^{-r}\sum_{i=0}^{n} B_i\,(1-2p)^i - (1-p)^n. \qquad (7)$$

Since we desire error detection codes with high code rates, k is generally much larger than r for practical error detection codes, and it is therefore much easier to determine the weight distribution of the 2^r code words in the dual code.

CRC codes are specialized cyclic codes where the generator polynomial has the form

$$g(x) = (x+1)\,p(x) \qquad (8)$$

where p(x) is a primitive irreducible polynomial of degree r − 1. Unshortened CRC codes have block lengths n, where $n = 2^{(r-1)} - 1$. The dual of a CRC code must then have a generator of the form

$$g_1(x) = \frac{x^n+1}{(x+1)\,p(x)}\ . \qquad (9)$$

Let

$$h_2(x) = p(x)\,,\ \text{and}\ g_2(x) = \frac{x^n+1}{p(x)} \qquad (10)$$

be the parity check and generator polynomials, respectively, of the dual of another code, C_2. Since

$$g_2(x) = (x+1)\,g_1(x), \qquad (11)$$

any code word in the dual of the code C_2, defined by $h_2(x)$ and $g_2(x)$, must be in the dual of the CRC code by (3). We obtain the code words in the dual of the unshortened CRC code by first obtaining all code words in the dual of C_2.

One way to generate the code words of a given cyclic code is through the use of a linear feedback shift register (LFSR), whose feedback is wired according to the coefficients of the parity-check polynomial for that code [5]. See Figure 1. An arbitrary code word is produced by initially loading the LFSR with a set of information digits, and then the n-tuple produced at its output when shifting the register is a code word. Since the output of the register is determined recursively from the initial loading of the register, the 2^{r-1} code words in the dual of C_2 are produced by considering all possible initial (r − 1)-tuple loadings of the shift register.

We make use of the fact that the parity check polynomial for C_2 is a primitive irreducible polynomial. Sequences produced by LFSRs whose feedback is determined by a primitive irreducible polynomial are known as *m-sequences*, and their usefulness in a variety of communications applications is well known. Certain properties of m-sequences have been established in the literature[6][7]. In particular, we note that a shift register, wired according to a primitive polynomial p(x) of degree r - 1 and initialized with some non-zero value, produces a sequence with period $2^{(r-1)} - 1$ [6][7]. Let

$$p(x) = \sum_{j=0}^{r-1} p_j x^j \quad , \tag{12}$$

be the primitive irreducible polynomial in question, and let the shift register be loaded with initial contents $\{s_{r-2}, s_{r-3}, \ldots, s_0\}$. Each time the shift register is clocked, it outputs m_i, the current value of x_0 and updates the shift register contents according to the current value of x_0 by the equations

$$\begin{cases} x_{r-2} = x_0 p_{r-1}, \\ x_i = x_{i+1} \oplus x_0 p_{i+1} \quad ; \quad i < r-2, \end{cases} \tag{13}$$

where the $\oplus$ indicates addition modulo 2. The output of the LFSR, being periodic, then satisfies

$$m_i = m_{i+2^{r-1}-1} \text{ for all i.} \tag{14}$$

If the initial contents of the shift register are all zero, then the shift register will produce the all zero code word. With any non-zero initial contents, the shift register produces a code word which is one cycle of the periodic output of the shift register. All cyclic shifts of the non-zero code word produced must be in the code by (4). Since this code word is one cycle of a sequence whose period is equal to the length of the code word, these distinct cyclic shifts account for $2^{(r-1)} - 1$ of the code words. Along with the all zero code word, this accounts for all $2^{(r-1)}$ code words corresponding to all $2^{(r-1)}$ possible initial loadings of the shift register. We may then consider the code words produced to be those that one would observe *by looking through a window*, that is, a shift register without feedback, *of width equal to the number of digits in a codeword* filled with the output of a LFSR. See Figure 2. We note that the number of ones in this window is the Hamming weight of a given code word in the dual of the code C_2. By counting the code words of various weights, we can calculate the probability of undetected error for the dual of the code C_2 using (6) or the code C_2 using (7). Let

$$w_i = \text{the number of code words of weight i in the dual of the code } C_2. \tag{15}$$

We note that all non-zero code words in the dual of the unshortened code C_2 must have the same weight, and by appealing to properties of m-sequences [6], we know that this weight is equal to

$$2^{r-2} - 1. \tag{16}$$

We now seek to enumerate the other code words in the dual of the unshortened CRC code. We consider the all one code word,

$$d(x) = \sum_{i=0}^{n-1} x^i \quad . \tag{17}$$

Since

$$h(x)d(x) = p(x)(x+1)\sum_{i=0}^{n-1} x^i = p(x)(x^n + 1) \equiv 0 \quad \text{modulo}\,(x^n + 1),$$

d(x), the all one code word, is in the dual of the CRC code. Since the dual of the CRC code is linear, the modulo 2 component sum of any two code words must also be a code word. When summing the all one code word with the code words in the dual of the unshortened code C_2, we complement the binary digits, creating code words of weight

$$\left(2^{r-1} - 1\right) - \left(2^{r-2} - 1\right) = 2^{r-2} \tag{18}$$

which must be distinct from the code words in the dual of the unshortened code C_2. We have therefore accounted for all 2^r code words in the dual of the unshortened CRC code.

If we let

$$b_i = \text{the number of code words of weight i in the dual of the CRC code,} \tag{19}$$

then

$$b_i = w_i + w_{n-i}, \tag{20}$$

where n is the block length of the code.

Since all unshortened CRC codes have the same weight structure regardless of the choice of the primitive irreducible polynomial of degree r – 1, they all perform identically with respect to their error detection capability. For unshortened CRC codes,

$$P_{ud}(p) = 2^{-r}\left(1 + (1-2p)^{2^{r-1}-1} + \left(2^{r-1}-1\right)(1-2p)^{2^{r-2}-1} + \left(2^{r-1}-1\right)(1-2p)^{2^{r-2}}\right) - (1-p)^{2^{r-1}-1}, \tag{21}$$

which monotonically increases for $0 \le p \le \frac{1}{2}$. P_{ud} may then be bounded by

$$P_{ud}\left(\frac{1}{2}\right) = 2^{-r} - (2)^{1-2^{r-1}} \le 2^{-r}. \tag{22}$$

However, the choice of primitive polynomial can be critical for shortened CRC codes. For some shortened CRC codes, P_{ud} does not monotonically increase for $0 \le p \le 1/2$ but rather displays peaks which can be several orders of magnitude larger than $P_{ud}(1/2)$. See, for example, Figure 11.

Shortening a code by x bits may be thought of as using x less bits of information by setting x of the information bits all equal to zero and using the same amount of redundancy. The shortened code can be thought of conceptually as noiselessly transmitting x all zero bits and the noisy transmitted message, and decoding as before after reinsertion of the deleted all zero bits. The code then becomes an (n – x, k – x) code.

In the dual of the shortened code CRC, the roles of information and redundancy are interchanged.
The dual code still has

$$(n-x)\ -(k-x)=n-k=r$$

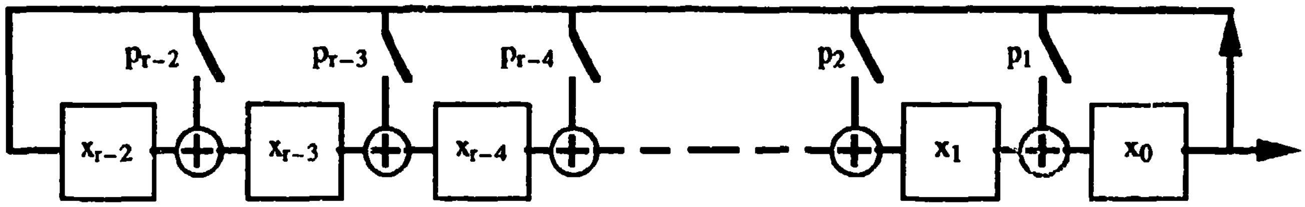
bits of information, but only k - x bits of redundancy. The shortened dual code may be thought of conceptually as having truncated redundancy bits. By conceptually aligning the shortened dual's truncated redundancy with the shortened code's all zero bits, orthogonality of the shortened code and its shortened dual is preserved.

As before, we can generate the code words in the dual of the shortened code C_2 using a LFSR as in Figure 1 by loading it with all possible combinations of r – 1 information bits, only now outputting n – x total bits. We therefore observe all code words through an appropriately down-sized window as in Figure 2.

As before, we obtain the weight distribution of the dual of the shortened CRC code from the dual of the shortened code C_2. The shortened all one code word of (17) is added to all code words of the dual of the shortened code C_2, and equations (19) and (20) still hold for the shortened block length. However, the weight structure of the dual codes is now irregular, and depends heavily on the choice of primitive polynomial. We therefore painstakingly produce all code words in the dual of the shortened code C_2, counting the number of code words of various weights produced. We may recursively account for all the code words in the dual of the shortened code C_2 as follows.

1. Initially, we have not seen any code words of any weight, so we set $w_i = 0$ for all i.
2. We account for the all zero code word by setting $w_0 = 1$.
3. We produce a code word by filling a window of length equal to the block length with the output of a linear feedback shift register LFSR1, counting the number of ones as they are inserted into the window.
4. $2^{r-1} - 1$ times, we
   a. account for the weight i of the current code word seen by incrementing w_i, and
   b. produce a new code word by shifting the registers of Figure 2, keeping track of its weight by noting the number of ones entering and exiting the window.

We can easily keep track of the input and output of ones from the window. The output of LFSR1 enters the window. If we note the initial state of LFSR1 before producing the first code word, then a second linear feedback shift register, LFSR2, initialized in the same way, will produce the sequence which is output from the window. See Figure 3. Since the weight enumeration of the dual of the shortened code C_2 is the time consuming part of the evaluation of a given CRC code, we have concentrated on optimizing the algorithm which produces the w_i.



Switch p_i is closed if the corresponding coefficient in $P(x) = \sum_{i=0}^{r-2} p_i x^i$ is non-zero.

Figure 1. Linear Feedback Shift Register (LFSR) for Operation Modulo P(x).

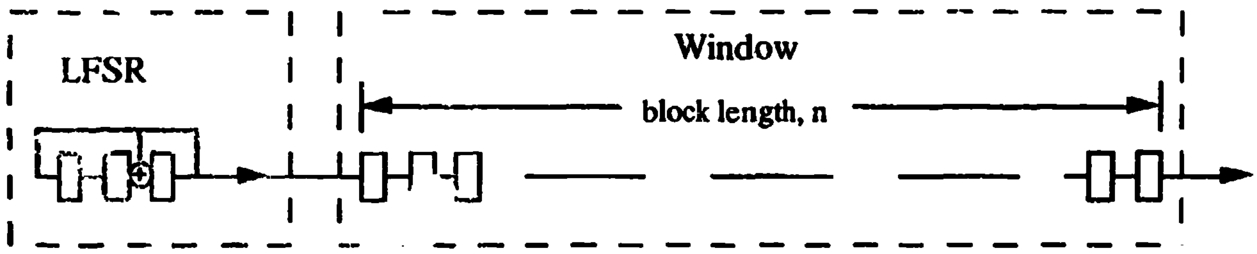
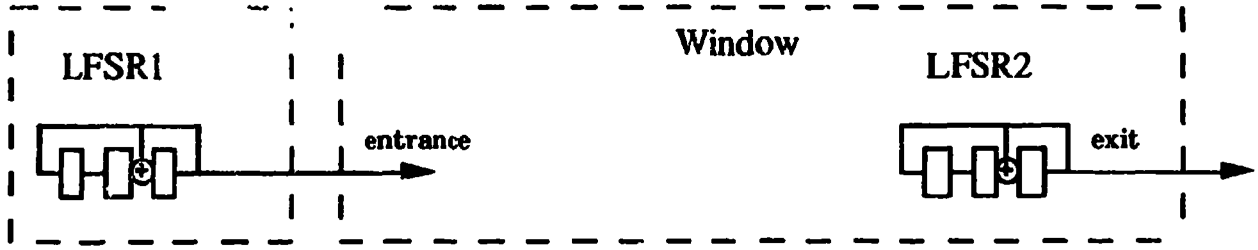


Figure 2. Code Word Viewer for Dual of Code C_2.



Initialize both LFSRs with the same non-zero contents and shift LFSR1 the block length, n, times. From then on, the outputs given by shifting LFSR1 and LFSR2 simultaneously gives the input and output of the window, respectively.

Figure 3. Code Weight Viewer for Dual of Code C_2.

III. PROGRAM FOR DETERMINING WEIGHT DISTRIBUTION AND ELIMINATING IMPROPER CRCS

This section describes a computer program that was written to aid in the evaluation of CRCs generated from various primitive polynomials. Most of the computer code that was written is in the C programming language. C is a relatively fast language that incorporates many of the useful bit operators needed in the computation of weight enumerators. Included at the end of this section is a listing of C code which calculates the weight enumerators, evaluates the probability of undetected error, $P_{ud}(p)$, formula (7) given in the section II, and eliminates those CRCs which exhibit a value of $P_{ud}(p)$ greater than $P_{ud}(1/2)$. CRCs which exhibit a value of $P_{ud}(p)$ for $p < 1/2$ greater than $P_{ud}(1/2)$ are called *improper*. In the next section, we provide 68000 assembler code for the critical loop which calculates the weight enumerators.

To streamline operation of the program with a minimum of user intervention, we designed this program to read a file of primitive polynomials, test CRCs based on them at a variety of consecutive block lengths, and write an output file of polynomials which were proper at the block lengths considered. The program prompts the user for the minimum probability of bit error to be considered, the number of values of the probability of bit error to be considered per decade, and the range of block lengths to be considered. The program uses log scaling of the probability of bit error, multiplying the current probability of bit error by a constant to get the next probability of bit error.

The program allocates memory to hold the w_i of the previous section. The polynomials under consideration are assumed to reside in a file called "polys" in the current directory. The main loop of the program reads the next polynomial in the file and processes it, and continues in this fashion until the end of the file is reached.

A polynomial is processed by first assuming that it has not been rejected, and assigning it to the variable "prim_poly". The degree of the polynomial is determined with the help of another variable called "poly". The weight enumerators of CRCs based on the polynomial are produced at each block length starting at the minimum block length and (7) is evaluated at all values of bit error probability considered, until either the polynomial is rejected or the maximum block length has been completed.

In producing the weight enumerators, it is convenient to represent the contents of the LFSR of Figure 1 of the previous section within the variable "shift0", with x_0 being the least significant bit, and the feedback in the variable "poly", with p_1 being the least significant bit. We note that the feedback is given by the primitive polynomial shifted one position toward the least significant bit, with the least significant bit discarded.

The w_i enumeration begins by initializing $w_i = 0$ for all $0 < i \leq n$ and $w_0 = 1$. To get the weight of the first code word, produced by the LFSR shift0 initialized with the value poly, we count the number of ones output by the LFSR in the first n shifts, where n is the block length. To test whether the output of the LFSR is a one and therefore if there is any feedback, we test to see if the least significant bit of shift0 is a one. If so, the weight is incremented and the next contents of the LFSR are obtained by shifting it one position and bit-wise XORing in the feedback, poly. If the least significant bit of shift0 is not a one, there is no feedback and no contribution to the weight, and the next contents of the LFSR are obtained by shifting it one position. We note that the initial value given to shift0 is the contents that it would contain after having the previous contents equal to 1. After producing the weight of the first code word, we increment the number w_i corresponding to this weight.

To produce the weights of the remainder of the code words, we now shift both LFSRs, shift0 and shift1, for $2^{r-1} - 2$ iterations. Shift1 is initialized with the same value, poly, that shift0 was originally initialized with, and its output represents bits exiting the window of Figure 3 of the previous section. The output of shift0 represents the input to the window. Whenever shift0 outputs a one, we increment the weight of the window, and whenever shift1 outputs a one, we decrement the weight of the window. After both LFSRs have shifted, we increment the number w_i corresponding to the weight of the current code word in the window. When the contents of shift1 equal 1, it has completed the desired number of iterations, a total of its sequence period – 1.

After completing the weight enumeration, we calculate $P_{ud}(p)$ using formula (7) of the previous section for each value of p from p = 1/2 down to the minimum value and compare $P_{ud}(p)$ to 1.0001 * $P_{ud}(1/2)$. If the CRC is improper, we print an appropriate message to the output file "stderr", and reject the polynomial. If the CRC is proper at all block lengths and values of p considered, we write it to the output file "stdout".

When evaluating formula (7), the first term is a polynomial in the variable (1-2p). We evaluate this polynomial in n multiplications and additions using Horner's rule [8].

III.1 (LISTING) C PROGRAM FOR ELIMINATING IMPROPER POLYNOMIALS

```
#include <stdio.h>
#include <math.h>
#include <malloc.h>

unsigned int *w;   /* jack's w */

main()
{
register int shift0,shift1;          /* begin and end of window  */
register unsigned int poly;          /* fast polynomial holders */
unsigned int ones;                   /* counter of ones in window */
int prim_poly;                       /* primitive read from file */
int poly_deg;                        /* degree of primitive, TBD */
int nperdecade;                      /* num of points to plot per decade */
double pe;                           /* prob bit error */
double pmin,pmax;                    /* min and max prob of bit error */
double logscale;                     /* 10 ^^ 1/nperdecade */
double pcalc;                        /* 1.0 - 2.0 * pe */
double pud;                          /* block prob of undetected error */
double bsum;                         /* holds sum of Bi * ((1-2p) ^^ i) */
int n;                               /* block length */
int min_block,max_block;             /* range of block lengths considered */
int reject;
int i;
double blkpudmax;

FILE *polyp,*fopen();
void * malloc();
double pow();
double log(),exp();

pmax = (double ) 0.5;

fprintf(stderr,"enter minimum probability of bit error Pe:");
fscanf(stdin,"%lf",&pmin);
fprintf(stderr,"enter points to consider per decade:");
fscanf(stdin,"%d",&nperdecade);
fprintf(stderr,"enter minimum size block:");
fscanf(stdin,"%d",&min_block);
fprintf(stderr,"enter maximum size block:");
fscanf(stdin,"%d",&max_block);
logscale = pow((double ) 10.0, ( (double ) 1.0 / (double ) nperdecade) );
fprintf(stderr,"logscale = %.6e\n",logscale);

fprintf(stdout,"Surviving polynomials for block size = %d to %d\n",min_block,max_block);

if ((w = (int *) malloc((max_block+1) * sizeof(int ))) == NULL)
        {
        fprintf(stderr,"sorry, couldn't malloc the w\n");
        exit(1);
        }
```

```
if ((polyp = fopen("polys","r")) == (FILE *) 0)
        {
        fprintf(stderr,"sorry, error opening polynomials file\n");
        exit(2);
        }

while (fscanf(polyp,"%o",&prim_poly) != EOF)
        {
        reject = 0;
        poly = prim_poly;

        /* determine degree of polynomial */
        for (poly_deg = -1; poly != 0 ; poly >>=1, poly_deg++) ;
        fprintf(stderr,"polynomial %o degree %d\n",prim_poly,poly_deg);

        poly = prim_poly>>1;

        for (n = min_block; !reject && n <= max_block; n++)
                {
                blkpudmax = pow((double)2.0, (double) (-poly_deg-1));
                blkpudmax -= pow((double) 0.5, (double ) n);

                w[0] = 1;
                for (i = 1; i <= n; i++)
                        w[i] = 0;
                /* count ones in initial pipeline */
                for (i=0, shift0 = poly, ones = 0; i<n ; i++)
                        {
                        if (shift0 & 1)
                                {
                                shift0 >>=1;
                                shift0 ^= poly;
                                ones++;
                                }
                        else shift0 >>=1;
                        }
                w[ones]++;
                /* run shift0 and shift1 for all but one of the m sequence */
                /* shift0's output enters pipeline */
                /* shift1's output exits pipeline */
                for (shift1 = poly; shift1 != 1 ; )
                        {
                        if (shift0 & 1)
                                {
                                shift0 >>=1;
                                shift0 ^= poly;
                                ones++;
                                }
                        else shift0 >>=1;
                        if (shift1 & 1)
                                {
                                shift1 >>=1;
                                shift1 ^= poly;
                                ones--;
                }
            else shift1 >>=1;
            w[ones]++;
            }
        fprintf(stderr,"%d\n",n);
        for (pe = pmax; pe >= pmin ; pe /= logscale)
            {
            pcalc = 1.0 - (2.0) * pe;
            /* calculate B sum via Horner's rule */
            bsum = (double ) (w[0] + w[n]); /* ie Bn */
            for (i = n - 1; i >= 0; i--)
                {
                bsum *= pcalc;
                bsum += ((double ) (w[i] + w[n-i]));
                }
            /* calculate pud */
            pud = bsum / pow(2.0, (double) (poly_deg+1));
            pud -= pow(1.0 - pe, (double ) n);
            if ((pud > 1.0e-13) && (pud > 1.00001 *blkpudmax))
                {
fprintf(stderr,"rejected %o at blocklength = %d pe = %.6e pud = %.6e\n",prim_poly,n,pe,pud);
                fprintf(stderr,"bounder for pud = %.6e\n",blkpudmax);
                reject = 1;
                }
            if (pud < 1.0e-13 || reject) break;
            }
        }
    if (n==max_block+1)
        fprintf(stdout,"%o\n",prim_poly);
    }
fclose(polyp);
free(w);
fprintf(stderr,"all done\n");
}
```

IV. ASSEMBLER CODE FOR CRITICAL SUBROUTINE

In this section, we describe code written in the Motorola 68000 microprocessor assembler language which improved execution time of the most time consuming part of the algorithm by a factor of 2. The assembler code is on the next page of this section. The time consuming part is the calculation of the w_i of section III. We optimized the instruction branching and the shifting of the LFSRs more than is possible in the C programming language.

We desired fast shifting of the LFSRs. In the C programming language, we must first test the least significant bit of a LFSR before shifting to determine the output and feedback operation. In assembler code, we can immediately shift the register and simultaneously perform an implicit test on its least significant bit, since a microprocessor condition code is set if the least significant bit was a one. Therefore, we follow the shift with a conditional branch and the code to perform had the bit been a one.

We also incorporate the initial zeroing of the w_i in the loop which produces the first code word. Since we count out a number equal to the block length in producing the first code word, it is natural to clear w_i as i is being counted. The initialization loop is displayed as a flow chart in Figure 4 of this section.

In Figure 5, we display a flow chart for the loop which enumerates all the w_i for $i > 0$. We improve the execution speed of the w address calculation by maintaining an address counter which increments or decrements by the number of bytes in the long integers used to represent the w_i.

At the beginning of the subroutine, we assume that to the global variable _poly contains the value of the feedback of the LFSRs, as in the previous section, and that the global variable _n contains the current block length under consideration.

IV.1 (LISTING) 68000 ASSEMBLER SUBROUTINE FOR DETERMINING WEIGHT ENUMERATORS

```
            public      _find_w
_find_w:
            link        a6,#.15
            movem.l     .16,-(sp)
lfsr1       equr        d7
lfsr2       equr        d6
numones     equr        d5
fdbkr       equr        d4
blkcnt      equr        a3
            move.l      #0,blkcnt
            move.l      blkcnt,numones
            move.l      _poly,fdbkr     ; put feedback in register
            move.l      fdbkr,lfsr1     ; set lfsr1 initially to second of sequence
            move.l      fdbkr,lfsr2     ; set lfsr2 initially to second of sequence
            move.l      _w,a0           ; put address of w in a0
nexti:
            move.l      blkcnt,d0       ; move block count to d0
            asl.l       #2,d0           ; multiply by 4 to get long's address offset
            clr.l       (a0,d0.l)       ; clear offset address
            lsr.l       #1,lfsr1        ; shift first LFSR
            bcc         addtoi          ; if no carry, we're done
            eor.l       fdbkr,lfsr1     ; otherwise, add in the feedback
            addq.l      #4,numones      ; and add one to the number of ones in window
addtoi:
            addq.l      #1,blkcnt       ; increment the block count
            cmp.l       _n,blkcnt       ; check to see if window is done
            bcs         nexti           ; if not, go around again
            move.l      _w,a0           ; put address of w in a0
            move.l      #1,(a0)         ; the all zero word conveniently goes here
wincrem:
            addi.l      #1,(a0,d5.l)    ; increment w[address]
            cmp.l       #1,lfsr2        ; check if done when lfsr2 = 1
            beq         findone         ; if so we're completely done here
            lsr.l       #1,lfsr1        ; in each iteration we enter window with output of lfsr1
            bcc         shift2nd        ; if no output, go shift LFSR2
            eor.l       fdbkr,lfsr1     ; else add in feedback,
            addq.l      #4,numones      ; one enters window
shift2nd:
            lsr.l       #1,lfsr2        ; shift of LFSR2
            bcc         wincrem         ; if done update count
            eor.l       fdbkr,lfsr2     ; otherwise add feedback,
            subq.l      #4,numones      ; one leaves window
            bra         wincrem         ; so enumerate unchanged count
findone:
            movem.l     (sp)+,.16
            unlk        a6
            rts
.15         equ         0
.16         reg         d4/d5/d6/d7/a3
```

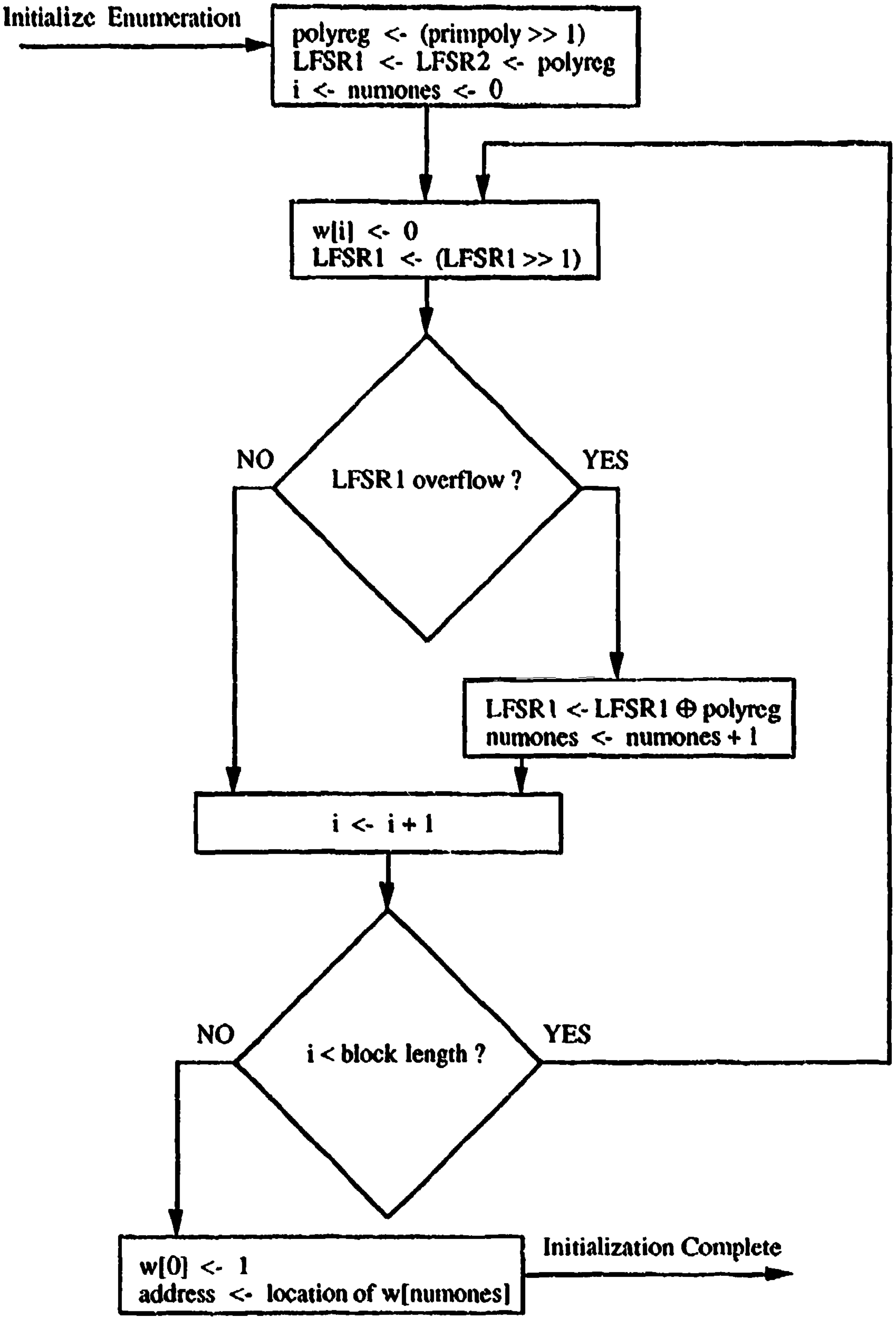


Figure 4. Initialization Loop of Assembler Subroutine Flowchart

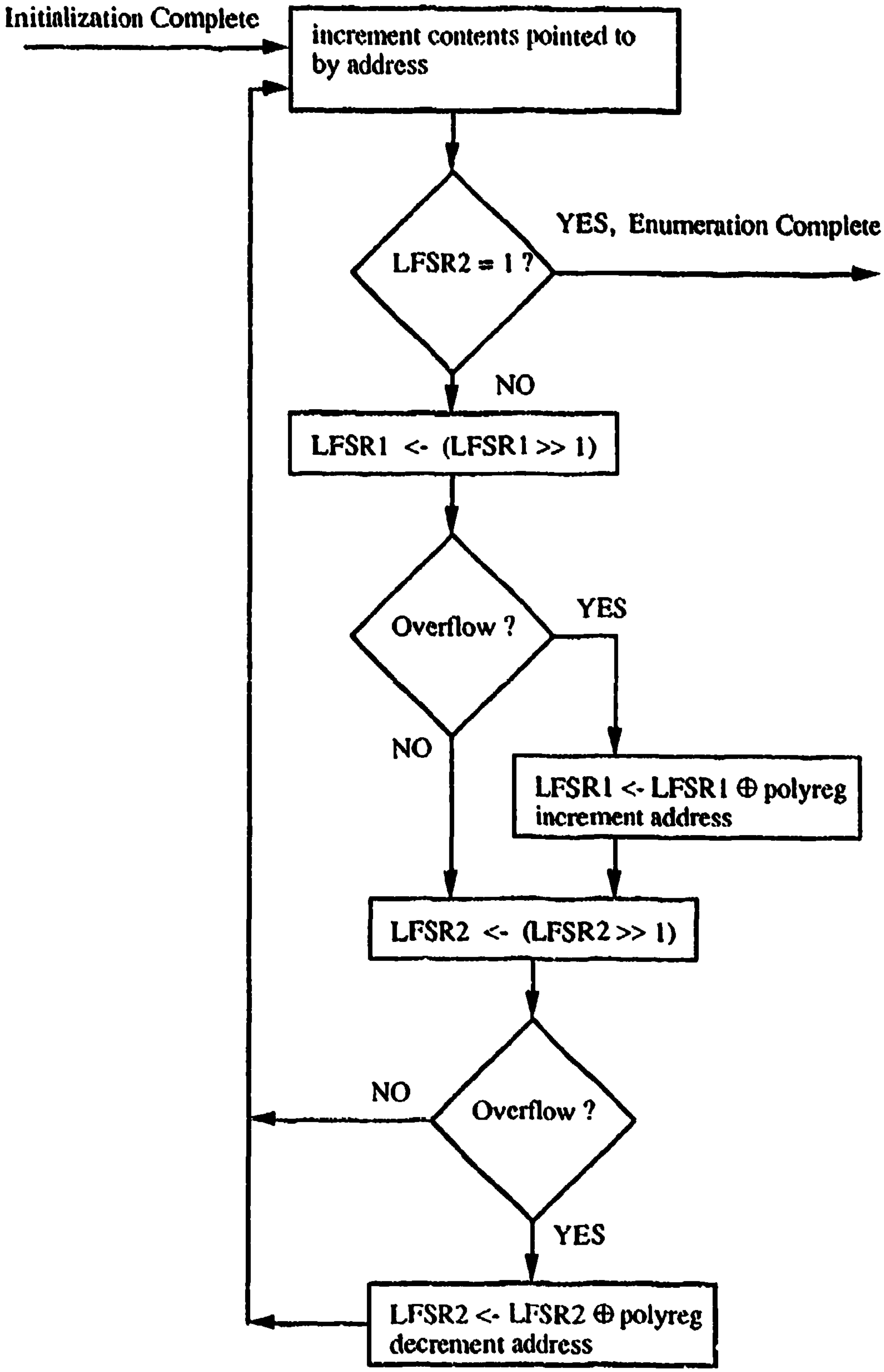


Figure 5. Main Loop of Assembler Subroutine Flowchart

V. SAMPLING THE WEIGHT DISTRIBUTION

As the number of CRC bits becomes large, the number of code words in the dual code, the number of potential polynomials, and the number of shortened block lengths all become large, prohibiting timely analysis of all the various possibilities. In this section, we investigate the possibility of extrapolating the performance of a given shortened CRC code by sampling the weight distribution of a subset of the code words. For CRCs with very many parity bits, this may be the only reasonable alternative until some closed form weight distribution formula becomes available. Unfortunately, our method seemed to give poor predictions of the code performance, even with relatively large sample sizes.

We note that because

$$b_i = w_i + w_{n-i}$$

the $\{b_i\}$ are symmetric about $i = n/2$, that is

$$b_i = b_{n-i}.$$

Therefore the true mean weight and any sample mean weight found by our method equals n/2. We sample a subset of the code words and hope to learn more about their weight distribution's higher order statistics. We conjecture that a distinguishing characteristic of improper CRCs lies in the "tail" of the distribution of the weights, and therefore we seek to compare the distribution of the tail of the weights to other known probability distributions.

As shown in the Figures of section IX on 24 bit CRCs, different primitive degree 23 polynomials can lead to differences on the order of 8 decades in error detection performance at critical block lengths and probabilities of bit error. Our sampling tests were performed on the 23 bit polynomials discussed in section IX, where we have plotted their performance. There are a total of 16,777,216 code words in each of the codes considered. We tried samples of 20,000 code words and could find no credible information supplied in the samples which would help to characterize the polynomials. We therefore increased our sample size to 2,000,000 code words, and still seemed to make poor predictions of the tail of the weight distribution.

We had hoped that the observed subset of code words would have roughly the same weight distribution as the complete set of code words. If so, we could take the observed number of code words of a given weight in the subset and multiply by a constant ratio to find the corresponding weight enumerator in the complete set.

The following charts compare the worst performing (40000041) and best performing (40435651) polynomials observed and true weight distributions at block length n = 32, where they showed dramatic differences in performance. We note that even with the unrealistically large percentage of total code words sampled, the tails of the distribution were represented inaccurately. In particular, we note that the samples displayed here tended to under estimate the number of code words of low weight in the worst code, and over estimate the number of code words of low weight in the best code. In Tables 1, we display similar charts for the other polynomials at block length n = 32, in order of their erroı detection performance, from worst to best. In Tables 2, we display charts for the worst and best polynomial at block length n = 1024. We note that our large sample failed to find virtually all of the code words of the lowest hundred weights for the worst code at block length n = 1024.

We also considered modelling the weight distribution as some known probability distribution, using higher order observed statistics such as sample variance as parameters to predict the distribution of the complete set. However, we found wide variations in the sample variances that we observed that did not correspond to performance measure. Table 3 compares the observed and true variance of the 24 bit CRCs at block length n = 32.

polynomial = 40000041
block length = 32

weight, i	b_i	sampled b_i	ratio
1	9	0	
2	47	0	
3	199	14	14.214286
4	718	44	16.318182
5	2241	247	9.072874
6	6293	744	8.458333
7	16495	2028	8.133629
8	41624	4761	8.742701
9	102189	11611	8.801051
10	236583	28279	8.366031
11	491619	59323	8.287157
12	887378	106157	8.359110
13	1379029	164485	8.383920
14	1858325	221242	8.399513
15	2202523	263142	8.370093
16	2326670	275848	8.434609

polynomial = 40435651
block length = 32

weight, i	b_i	sampled b_i	ratio
2	1	0	
3	25	1	25.000000
4	122	19	6.421053
5	761	92	8.271739
6	3603	515	6.996117
7	13146	1757	7.482072
8	40992	5343	7.672094
9	109850	13415	8.188595
10	252225	30603	8.241839
11	503127	60515	8.314087
12	881190	105912	8.320020
13	1358135	162523	8.356571
14	1843371	216773	8.503693
15	2209260	262025	8.431486
16	2345598	281016	8.346849

Table 1. Sampled Weight Distributions for 24 Bit CRCs at Block Length n = 32

	Weight = i	B_i	Sampled B_i	Ratio (B_i/Sampled B_i)
polynomial = 40404041	2	31	0	
	3	19	0	
	4	424	46	9.217391
	5	575	58	9.913793
	6	6349	833	7.621849
	7	12146	1651	7.356753
	8	47756	5859	8.150879
	9	110274	12947	8.517340
	10	243679	28869	8.440854
	11	506553	60946	8.311505
	12	851672	102356	8.320685
	13	1358229	162058	8.381129
	14	1849141	218780	8.452057
	15	2206508	262636	8.401392
	16	2390502	285924	8.360620
polynomial = 40000063	1	9	0	
	2	38	0	
	3	118	16	7.375000
	4	377	56	6.732143
	5	1296	164	7.902439
	6	4286	455	9.419780
	7	13285	1654	8.032044
	8	38734	4630	8.365875
	9	103651	12243	8.466144
	10	245858	29491	8.336713
	11	505668	59811	8.454431
	12	896375	106366	8.427270
	13	1376086	163966	8.392508
	14	1847994	220876	8.366658
	15	2194191	261953	8.376277
	16	2321282	276640	8.390985
polynomial = 50000241	1	5	5	1.000000
	2	19	9	2.111111
	3	65	13	5.000000
	4	241	29	8.310345
	5	939	121	7.760331
	6	3709	462	8.028139
	7	13143	1602	8.204120
	8	40222	4729	8.505392
	9	107645	12603	8.541220
	10	250391	29820	8.396747
	11	504537	59851	8.429884
	12	886271	105534	8.397967
	13	1363515	162719	8.379568
	14	1844057	219930	8.384745
	15	2204455	263178	8.376289
	16	2338786	278792	8.389000

	Weight = i	B_i	Sampled B_i	Ratio (B_i/Sampled B_i)
polynomial = 40010061	1	2	0	
	2	14	0	
	3	46	0	
	4	204	19	10.736842
	5	806	114	7.070175
	6	3202	402	7.965174
	7	12554	1542	8.141375
	8	40788	4582	8.901790
	9	110550	12818	8.624590
	10	254886	30687	8.305993
	11	505210	60211	8.390660
	12	878196	104756	8.383253
	13	1352434	161143	8.392757
	14	1841098	218790	8.414909
	15	2212702	264280	8.372567
	16	2351830	281314	8.360160
polynomial = 40006341	1	3	0	
	2	9	0	
	3	26	7	3.714286
	4	146	23	6.347826
	5	832	94	8.851064
	6	3483	384	9.070313
	7	12735	1356	9.391593
	8	40752	4540	8.976211
	9	110025	13105	8.395651
	10	253305	29983	8.448287
	11	505484	60124	8.407358
	12	881422	104546	8.430949
	13	1352954	160245	8.443034
	14	1838307	218754	8.403535
	15	2212245	265185	8.342270
	16	2353758	283310	8.308065
polynomial = 40220151	2	7	0	
	3	24	0	
	4	130	28	4.642857
	5	824	119	6.924370
	6	3501	389	9.000000
	7	13008	1600	8.130000
	8	41088	5311	7.736396
	9	109136	13489	8.090741
	10	252367	30357	8.313305
	11	506632	60451	8.380870
	12	881502	104344	8.448037
	13	1352360	161444	8.376651
	14	1841277	218556	8.424738
	15	2212320	263211	8.405120
	16	2348862	281404	8.346939

	Weight = i	B_i	Sampled B_i	Ratio (B_i/Sampled B_i)
polynomial = 41224445	2	3	0	
	3	28	0	
	4	131	3	43.666667
	5	796	62	12.838710
	6	3573	388	9.208763
	7	12952	1538	8.421326
	8	40898	4928	8.299107
	9	109976	13331	8.249644
	10	252431	29911	8.439404
	11	503972	59599	8.456048
	12	881997	105479	8.361826
	13	1356196	161509	8.397031
	14	1840121	219918	8.367305
	15	2210384	263114	8.400860
	16	2350298	280442	8.380692
polynomial = 40405463	2	9	0	
	3	10	0	
	4	129	17	7.588235
	5	778	99	7.858586
	6	3567	455	7.839560
	7	13156	1507	8.729927
	8	41086	4713	8.717590
	9	109540	13044	8.397731
	10	251645	30186	8.336480
	11	504982	60590	8.334412
	12	882287	105141	8.391465
	13	1354390	161634	8.379363
	14	1842955	219919	8.380154
	15	2211448	263101	8.405320
	16	2345250	279190	8.400193
polynomial = 40103271	2	6	0	
	3	14	0	
	4	121	8	15.125000
	5	774	93	8.322581
	6	3574	413	8.653753
	7	13332	1619	8.234713
	8	40926	4875	8.395077
	9	109108	12963	8.416879
	10	252410	29694	8.500370
	11	504362	59445	8.484515
	12	881975	105440	8.364710
	13	1357058	162036	8.375040
	14	1840138	219690	8.376066
	15	2209656	263612	8.382228
	16	2350306	280226	8.387180

Table 2. Sampled Weight Distributions for 24 Bit CRCs at Block Length n = 1024

	Weight = i	B_i	Sampled B_i	Ratio (B_i/Sampled B_i)
polynomial = 40000041	335	5	0	
	336	23	0	
	337	17	0	
	338	24	0	
	339	14	0	
	340	11	0	
	341	13	0	
	342	23	0	
	343	22	0	
	344	19	0	
	345	20	0	
	346	8	0	
	347	5	0	
	348	9	0	
	349	13	0	
	350	6	0	
	351	8	0	
	352	7	0	
	353	4	0	
	354	7	0	
	355	12	0	
	356	6	0	
	357	6	0	
	358	12	0	
	359	8	0	
	360	5	0	
	361	7	0	
	362	4	0	
	363	7	0	
	364	14	0	
	365	7	0	
	366	15	0	
	367	34	0	
	368	35	0	
	369	31	0	
	370	30	0	
	371	39	0	
	372	20	0	
	373	29	0	
	374	25	0	
	375	14	0	
	376	5	0	
	377	7	0	
	378	3	0	
	379	3	0	
	380	6	0	
	381	4	0	
	382	9	0	
	383	18	0	
	384	24	0	
	385	8	0	
	386	4	0	
	387	4	0	
	388	3	0	
	389	4	0	
	390	14	0	
	391	9	0	
	392	7	0	
	393	6	0	

	Weight = i	B_i	Sampled B_i	Ratio (B_i/Sampled B_i)
polynomial = 40000041	394	8	0	
	395	3	0	
	396	2	0	
	397	21	0	
	398	16	0	
	399	6	0	
	400	3	0	
	401	6	0	
	402	6	0	
	403	5	0	
	404	10	0	
	405	14	0	
	406	24	0	
	407	32	0	
	408	47	0	
	409	46	0	
	410	53	0	
	411	51	0	
	412	84	0	
	413	90	0	
	414	97	0	
	415	98	0	
	416	143	0	
	417	190	0	
	418	186	0	
	419	145	0	
	420	165	0	
	421	180	0	
	422	183	0	
	423	251	0	
	424	316	0	
	425	311	0	
	426	332	0	
	427	326	0	
	428	347	0	
	429	375	0	
	430	364	0	
	431	408	0	
	432	503	0	
	433	514	0	
	434	498	2	249.000000
	435	548	17	32.235294
	436	567	10	56.700000
	437	650	8	81.250000
	438	689	15	45.933333
	439	741	31	23.903226
	440	787	34	23.147059
	441	903	51	17.705882
	442	926	28	33.071429
	443	1062	51	20.823529
	444	1222	76	16.078947
	445	1366	109	12.532110
	446	1522	121	12.578512
	447	1548	127	12.188976
	448	1596	112	14.250000
	449	1649	141	11.695035
	450	1977	177	11.169492
	451	2166	208	10.413462
	452	2306	192	12.010417
	453	2377	181	13.132597
	454	2364	178	13.280899

	Weight = i	B_i	Sampled B_i	Ratio (B_i/Sampled B_i)
polynomial = 40000041	455	2642	170	15.541176
	456	2707	195	13.882051
	457	2929	260	11.265385
	458	3239	340	9.526471
	459	3604	405	8.898765
	460	3888	447	8.697987
	461	4519	561	8.055258
	462	4901	585	8.377778
	463	5664	627	9.033493
	464	6124	653	9.378254
	465	6886	667	10.323838
	466	7673	809	9.484549
	467	8845	973	9.090442
	468	9665	1060	9.117925
	469	10913	1271	8.586153
	470	12350	1478	8.355886
	471	13223	1458	9.069273
	472	14280	1512	9.444444
	473	15859	1795	8.835097
	474	17878	2169	8.242508
	475	21010	2618	8.025210
	476	24427	3169	7.708110
	477	27631	3565	7.750631
	478	31546	3881	8.128317
	479	35932	4356	8.248852
	480	41384	4838	8.553948
	481	46773	5395	8.669694
	482	54250	6001	9.040160
	483	61639	7058	8.733211
	484	70219	7983	8.796067
	485	81368	9638	8.442415
	486	93432	11558	8.083752
	487	106779	13213	8.081359
	488	121354	14963	8.110272
	489	136433	16966	8.041554
	490	152584	18870	8.086063
	491	168140	20965	8.020033
	492	185471	23025	8.055201
	493	202442	25336	7.990290
	494	220183	27422	8.029429
	495	237707	29508	8.055680
	496	256835	31222	8.226091
	497	274892	33545	8.194724
	498	292629	36481	8.021408
	499	311050	38306	8.120138
	500	326600	39762	8.213873
	501	344840	41364	8.336718
	502	360291	42596	8.458329
	503	371886	44376	8.380341
	504	384627	45530	8.447771
	505	398836	47277	8.436153
	506	407794	48032	8.490048
	507	416347	48932	8.508686
	508	422590	49351	8.562947
	509	425809	49440	8.612642
	510	429997	49307	8.720810
	511	433246	49787	8.701990
	512	435328	50122	8.685368

	Weight = i	B_i	Sampled B_i	Ratio (B_i/Sampled B_i)
polynomial = 40435651	446	2	0	
	447	3	0	
	448	9	3	3.000000
	449	42	19	2.210526
	450	98	47	2.085106
	451	168	60	2.800000
	452	198	39	5.076923
	453	349	39	8.948718
	454	440	46	9.565217
	455	528	53	9.962264
	456	809	56	14.446429
	457	992	88	11.272727
	458	1145	127	9.015748
	459	1416	147	9.632653
	460	1900	201	9.452736
	461	2285	264	8.655303
	462	2851	264	10.799242
	463	3606	398	9.060302
	464	4419	510	8.664706
	465	5356	589	9.093379
	466	6485	654	9.915902
	467	7890	903	8.737542
	468	9775	1199	8.152627
	469	11678	1434	8.143654
	470	13643	1794	7.604794
	471	15899	2086	7.621764
	472	18580	2258	8.228521
	473	21610	2685	8.048417
	474	25001	3144	7.951972
	475	28624	3537	8.092734
	476	33089	4097	8.076397
	477	37711	4452	8.470575
	478	43851	5322	8.239572
	479	50211	6381	7.868829
	480	56689	7279	7.788020
	481	64526	8463	7.624483
	482	73592	9733	7.561081
	483	82756	10670	7.755951
	484	92279	11460	8.052269
	485	103536	12922	8.012382
	486	112560	14057	8.007398
	487	123787	15282	8.100183
	488	135715	16700	8.126647
	489	150118	18543	8.095670
	490	164676	20180	8.160357
	491	178776	22022	8.118064
	492	192944	23804	8.105528
	493	209033	25506	8.195444
	494	224106	27419	8.173383
	495	238930	29136	8.200508
	496	253153	30880	8.197960
	497	267167	32223	8.291190
	498	283347	34514	8.209625
	499	299062	36358	8.225480
	500	317179	38524	8.233283
	501	331770	39890	8.317122
	502	344763	41180	8.372098
	503	355956	41854	8.504707
	504	367040	42898	8.556110
	505	377846	44339	8.521753
	506	387904	45567	8.512827

	Weight = i	B_i	Sampled B_i	Ratio (B_i/Sampled B_i)
polynomial = 40435651	507	396527	46034	8.613785
	508	402684	46210	8.714218
	509	408547	46369	8.810779
	510	412688	46336	8.906423
	511	417599	47157	8.855504
	512	421378	47192	8.929013

Table 3. Variance of Sample Weight Distributions for 24 Bit CRCs at Block Length n = 32

	Sample Variances		
	Sample Size = 20000	Sample Size = 2000000	ALL
Polynomial			
40000041	15.335	15.958	16.0
40404041	15.005	16.074	16.0
40000063	16.583	15.973	16.0
50000241	18.394	15.972	16.0
40010061	15.981	15.957	16.0
40006341	15.903	15.869	16.0
40220151	16.366	16.104	16.0
41224445	15.151	15.981	16.0
40405463	16.068	16.008	16.0
40103271	16.415	15.950	16.0
40435651	15.202	16.216	16.0

VI. HARDWARE TESTER FOR CRC CODES

The sampling results of the previous section convince us that to properly evaluate a given CRC code we have to enumerate the weights of all code words. In order to reasonably evaluate a larger number of 32 bit and larger CRCs at a variety of moderate block lengths, we estimate that we must increase our weight enumeration capability by at least a factor of ten. We therefore sought a cost effective alternative to the purchase of ten times the computers and associated labor hours involved.

In this section, we informally propose specialized hardware that would efficiently determine the weight enumerators of a CRC code at moderate block lengths. We intend to formalize this proposal for a Phase Two continuation of the project.

The optimized assembler code of section IV gives us a good estimate of the number of machine cycles required by a particular general purpose microprocessor to produce the weight enumerators. The main loop requires approximately 25 machine cycles per iteration (68030, cache case). By producing specialized hardware that would require only 1 machine cycle per iteration, we should be able to gain a factor of 25 in execution time for the same cycle frequency. Additionally, if we produce hardware in a faster logic family than CMOS, such as ECL, there should be additional gains to be made.

Figure 6 is a block diagram of the contemplated hardware. We envision a personal computer as the interface between the user and the specialized weight enumerator circuitry. A controller card in the PC would enable it to download the parameters of the code to be analyzed to the specialized hardware, which would be implemented in ECL. The programmable shift registers would play the role of lfsr1 and lfsr2 in the assembly code. Two counters would be employed to count out the n bits of the first code word and accumulate its weight. The main loop of the algorithm would involve shifting both registers, producing the next weight, and updating a weight enumerator in RAM once per cycle. A relatively simple state machine whose modes of operation are shown in Figure 7 would control the weight enumeration without intervention from the controller. When the weight enumeration is complete, the ECL circuitry would alert the controller to collect the weight enumerators and issue the parameters of the next code. While the weight enumeration of the next code proceeds, the personal computer would be free to (comparatively slowly) evaluate the error detection capability of the given code.

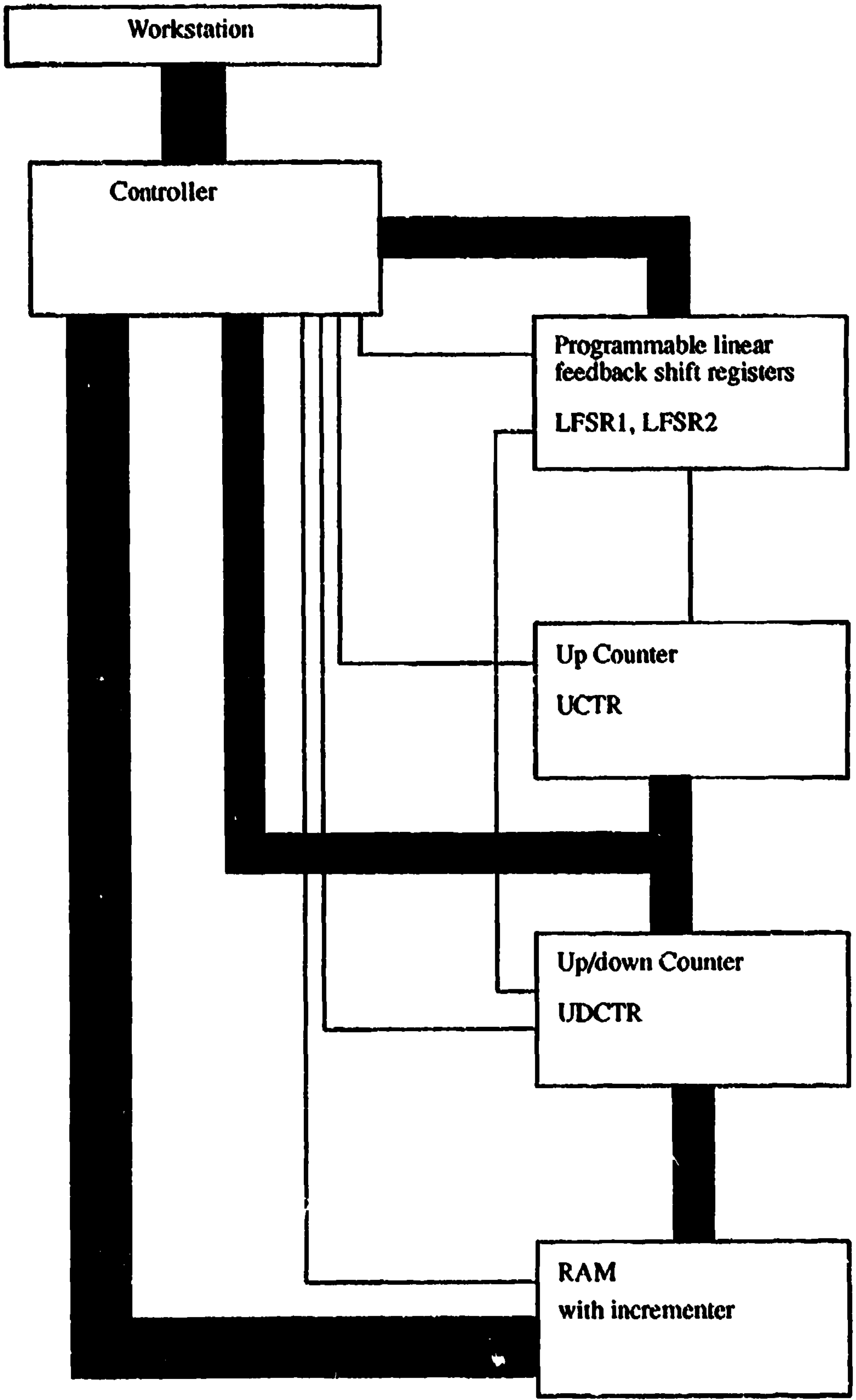
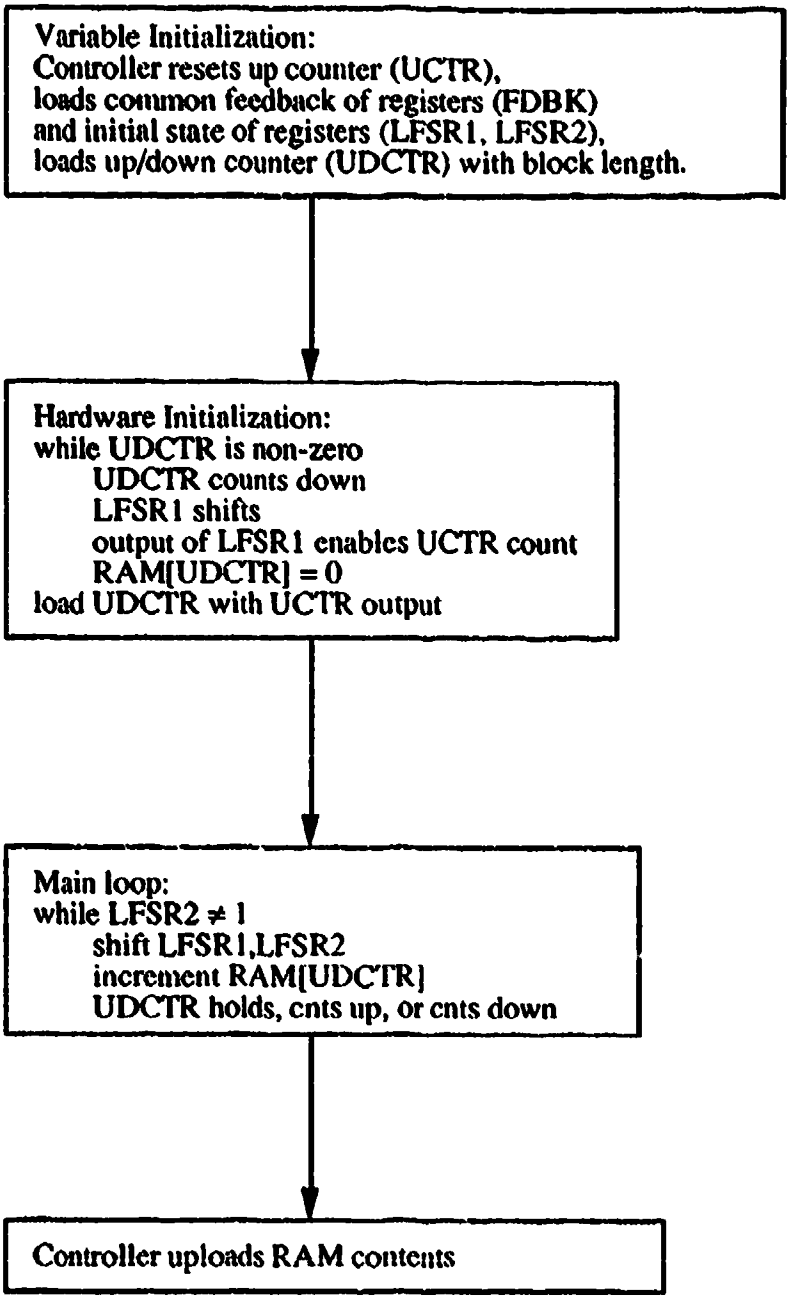


Figure 6. Contemplated Hardware



Variable Initialization:
Controller resets up counter (UCTR),
loads common feedback of registers (FDBK)
and initial state of registers (LFSR1, LFSR2),
loads up/down counter (UDCTR) with block length.

```
Hardware Initialization:
while UDCTR is non-zero
    UDCTR counts down
    LFSR1 shifts
    output of LFSR1 enables UCTR count
    RAM[UDCTR] = 0
load UDCTR with UCTR output
```

```
Main loop:
while LFSR2 ≠ 1
    shift LFSR1,LFSR2
    increment RAM[UDCTR]
    UDCTR holds, cnts up, or cnts down
```

Controller uploads RAM contents

Figure 7. Hardware Modes of Operation

VII. EVALUATION OF 8 BIT CRCS

We used Peterson and Weldon [5] as a reference which supplies all primitive irreducible polynomials of degree ≤ 16. With CRCs of the form (x + 1) p(x), a given primitive irreducible polynomial p(x) and its (primitive irreducible) reciprocal polynomial perform identically as error detection codes. We therefore test and list, as in [5], only one of any pair of a polynomial and its reciprocal. We list the polynomials, as in [5], using octal notation. For example, the notation

3525

in octal is converted to binary

011 101 010 101

and represents the polynomial

$$p(x) = x^{10} + x^9 + x^8 + x^6 + x^4 + x^2 + 1.$$

The reciprocal polynomial is 2527.

8 bit CRCs based on the nine primitive irreducible polynomials in [5] were tested at all block lengths from n = 9 to n = 127. At each block length, the probability of undetected error was evaluated for twenty logarithmically spaced values in each decade for p, from $p = 10^{-6}$ to $p = 1/2$. Any polynomials found to be improper at any of these values was rejected.

Degree 7 polynomials p(x) which produce proper 8 bit CRCs of the form (x + 1) p(x) over the range of all block lengths considered are

211 235 325 313 345,

and their reciprocal polynomials.

VIII. EVALUATION OF 16 BIT CRCS

As in section VII, we started with an exhaustive list of nearly 1000 primitive polynomials of degree 15 from Peterson and Weldon. 16 bit CRCs based on the primitive irreducible polynomials in [5] were tested at all block lengths from n = 17 to n = 32767. At each block length, the probability of und^etected error was evaluated for twenty logarithmically spaced values in each decade for p, from $p = 10^{-6}$ to $p = 1/2$. Any polynomials found to be improper at any of these values was rejected.

Degree 15 polynomials p(x) which produce proper 16 bit CRCs of the form (x + 1) p(x) over the range of all block lengths considered are

103451
112273
151043
165033
132103
115307
165113
134325
132127
175515
115537
133113
155303
126155
104111
152351
125537
154155
162241
171125
144225
106633
116675
105071
114231
134165
111267
112611
114273
153143
110427
165355
157241
135751
124647
135267
132507
125471
161205
141115
141445
123735
145433
154507
142751
101551
133011
101515
116645
111243
154233
106445
115373
165633
115155
123023
172213
131211
150243
166541
101661
113645
146727
102615
106251
132357
144425
126711
135443
134435
155027
121553
146705
165565
166267
104427
163273
166113
133257
166653
130745
102561
160521
105213
160511
141655
102513
113373
156635
154545
105713
106611
137061
141231
107645
162315
116631
105143
121305
117243
127071
117131
115271
102265
115667
110165
144467
131623,
134531
132367
120447
153731
124243
113255
141151
150633
133553
134241
102471
112347
143271
163365
146155
154515
164313
170325
134205
127457
144151
142457
112553
122123
163555
135523
142305
156333
117511
144275
147321
133571
130305
146025
171131
115523
162455
165535
120463
164155
121641
171115
121327
135565
161465
165303
150327
156321
167331
164453
105237
131367
103145
127143
115141
134447
150225
112407
121355
111423
112365
164447
162153
146753
110435
110405
131667
110255
124335
155725
144713
105415
163123
175651
164561
155335
130635
143227

and their reciprocals.

We note that most polynomials were rejected at relatively short block lengths (< n = 250), and that only one polynomial was rejected in the range 1000 ≤ n ≤ 2000. No polynomials which survived to n = 2000 were later rejected. We also note that neither of the standard 16 bit CRCs, CRC-16 generated from 100003 nor CRC-CCITT generated from 170037, is proper at all block lengths.

In Figures 8, 9, and 10 we plot the error detection performance of 16 bit CRCs based on four of the primitive polynomials from the chart at block lengths 256, 1024, and 4096, respectively. The performance of all four codes was nearly identical.

In all the graphs, we reference a given plot with the notation "xxxx"-"yyy", where xxxx is the octal representation of the primitive polynomial p(x) used to produce a CRC of the form (x + 1) p(x), and yyy is the block length n under consideration.

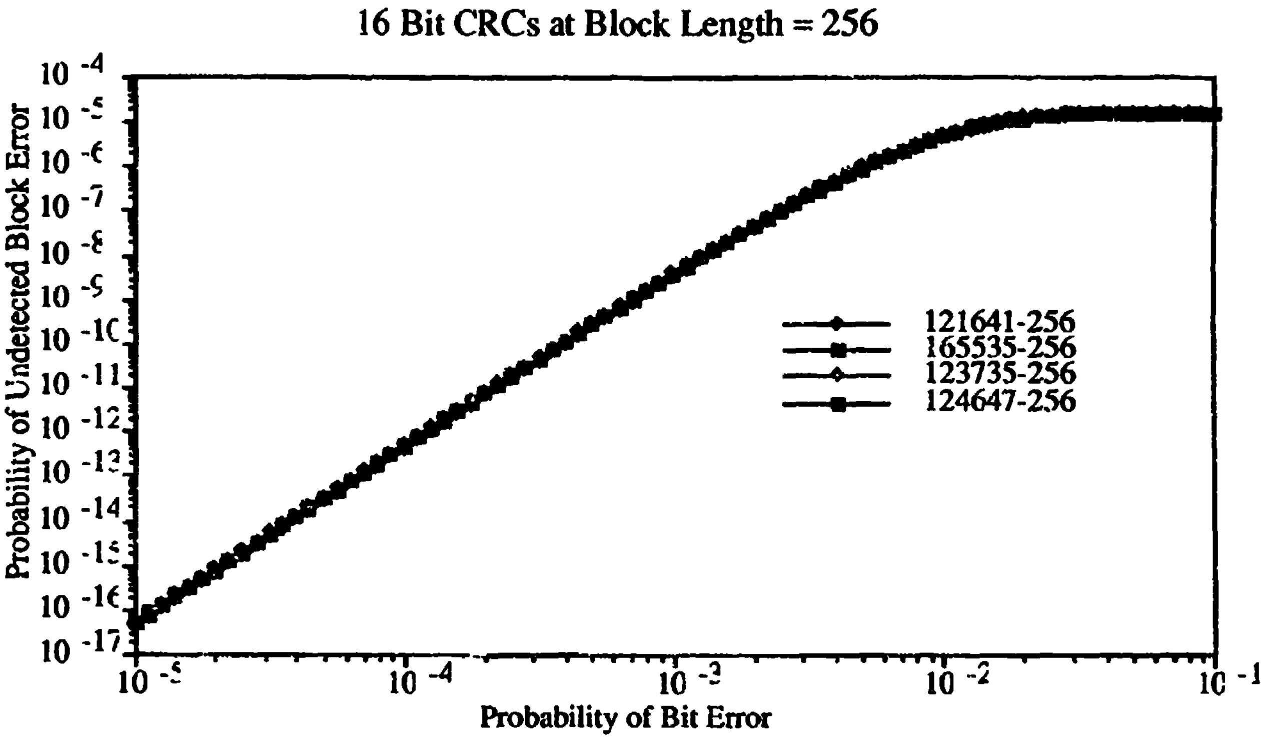


Figure 8. Probability of undetected block error vs. probability of bit error for 16 bit CRCs at block length n = 256.

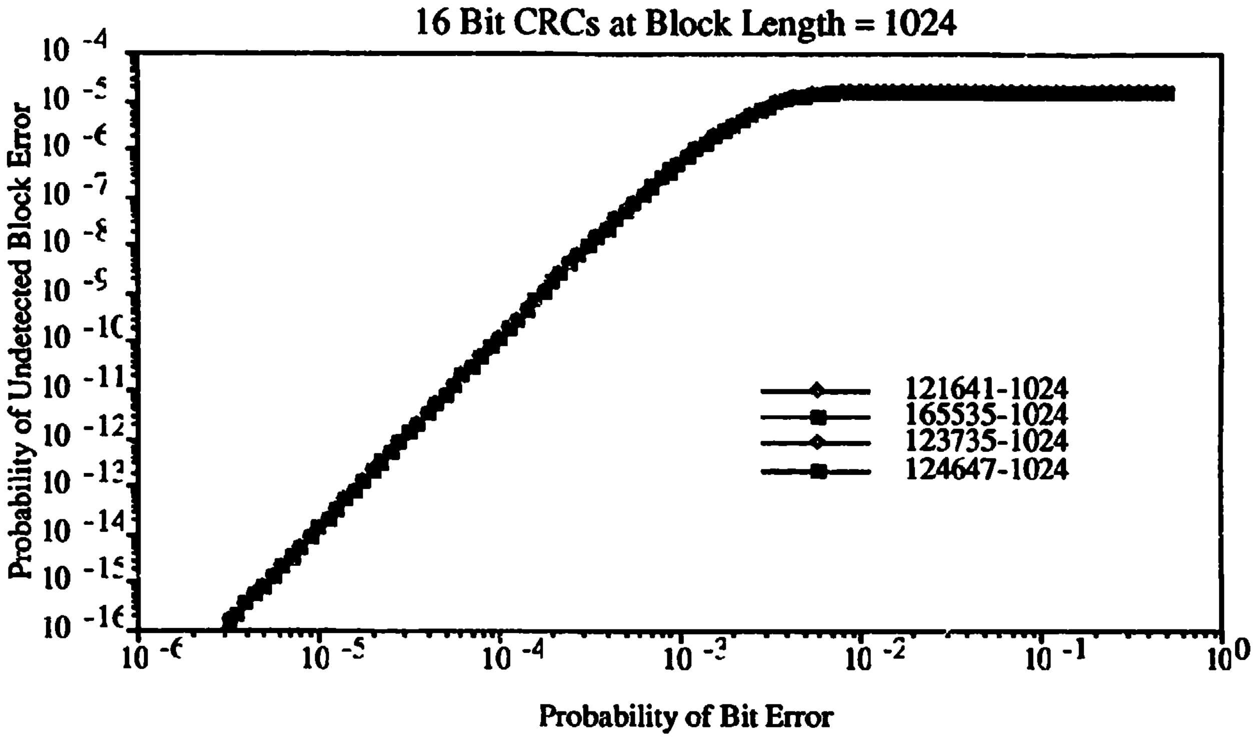


Figure 9. Probability of undetected block error vs. probability of bit error for 16 bit CRCs at block length n = 1024.

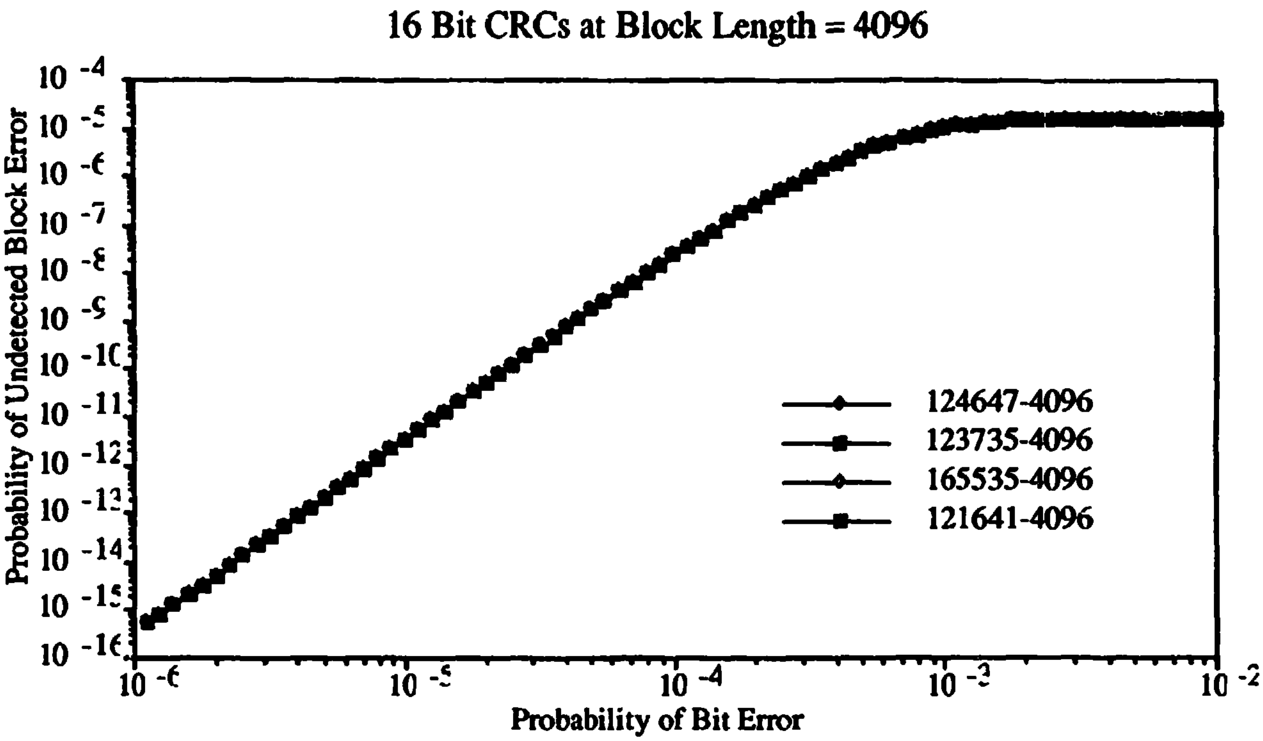


Figure 10. Probability of undetected block error vs. probability of bit error for 16 bit CRCs at block length n = 4096.

IX. EVALUATION OF 24 BIT CRCS

As in section VIII, we started with the list of primitive polynomials of degree 23 from Peterson and Weldon. The list contains a total of 11 polynomials. 24 bit CRCs based on the primitive irreducible polynomials in [5] were tested at all block lengths from n = 25 to n = 4000. At each block length, the probability of undetected error was evaluated for twenty logarithmically spaced values in each decade for p, from $p = 10^{-6}$ to $p = 1/2$.

We were immediately able to reject all but one of the polynomials as improper at short block lengths, as is demonstrated in Figure 11. The remaining polynomial, generated from 40435651, was tested extensively, since we propose this polynomial to generate a new standard CRC-24Q. The polynomial was tested at all blocks lengths up to 4000, and at block lengths 4096, 8192, 16384, and 32767. The polynomial displayed a very slight rise in $P_{ud}(p)$ that was less than 0.1% above the value of $P_{ud}(1/2)$ at a few block lengths, which we find to be very minor and unworthy of rejection.

In Figures 11 through 14, we have plotted the probability of undetected error versus the probability of bit error for all 24 bit CRCs generated from the polynomials of [5], at block lengths 32, 256, 1024, and 4096, respectively. We note that in the first two figures, the dramatic difference in error detection performance of shortened CRCs is depicted. We also note that it is insufficient to predict the performance of a given shortened CRC by sampling its performance at a given block length, such as 4096, where all the polynomials appear to perform nearly identically.

In Figure 15, we have plotted the performance of the new proposed standard CRC-24Q, at a variety of blocklengths up to 32,767. We note its nearly uniform performance over this range with respect to propriety.

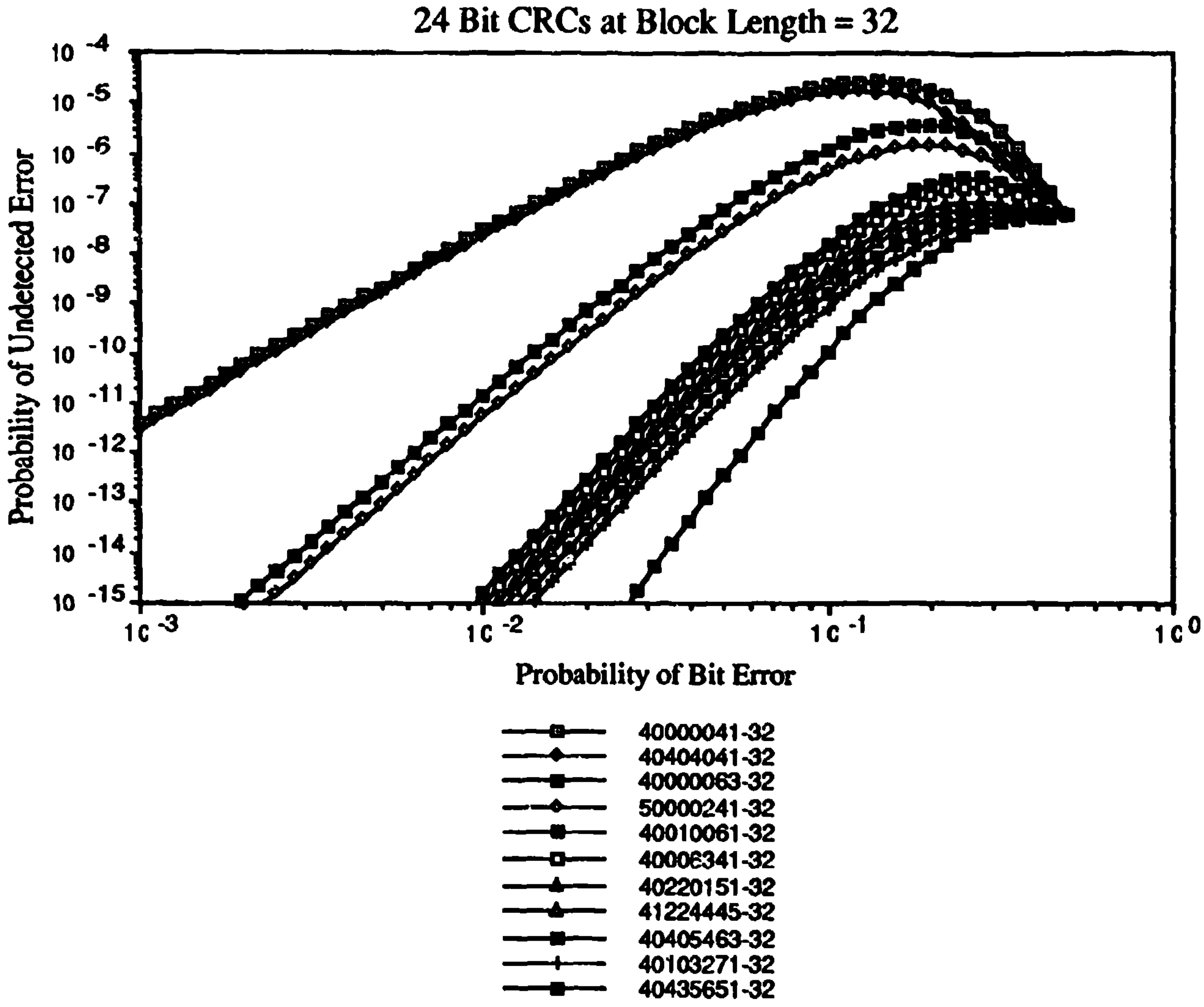


Figure 11. Probability of undetected block error vs. probability of bit error for 24 bit CRCs at block length n = 32.

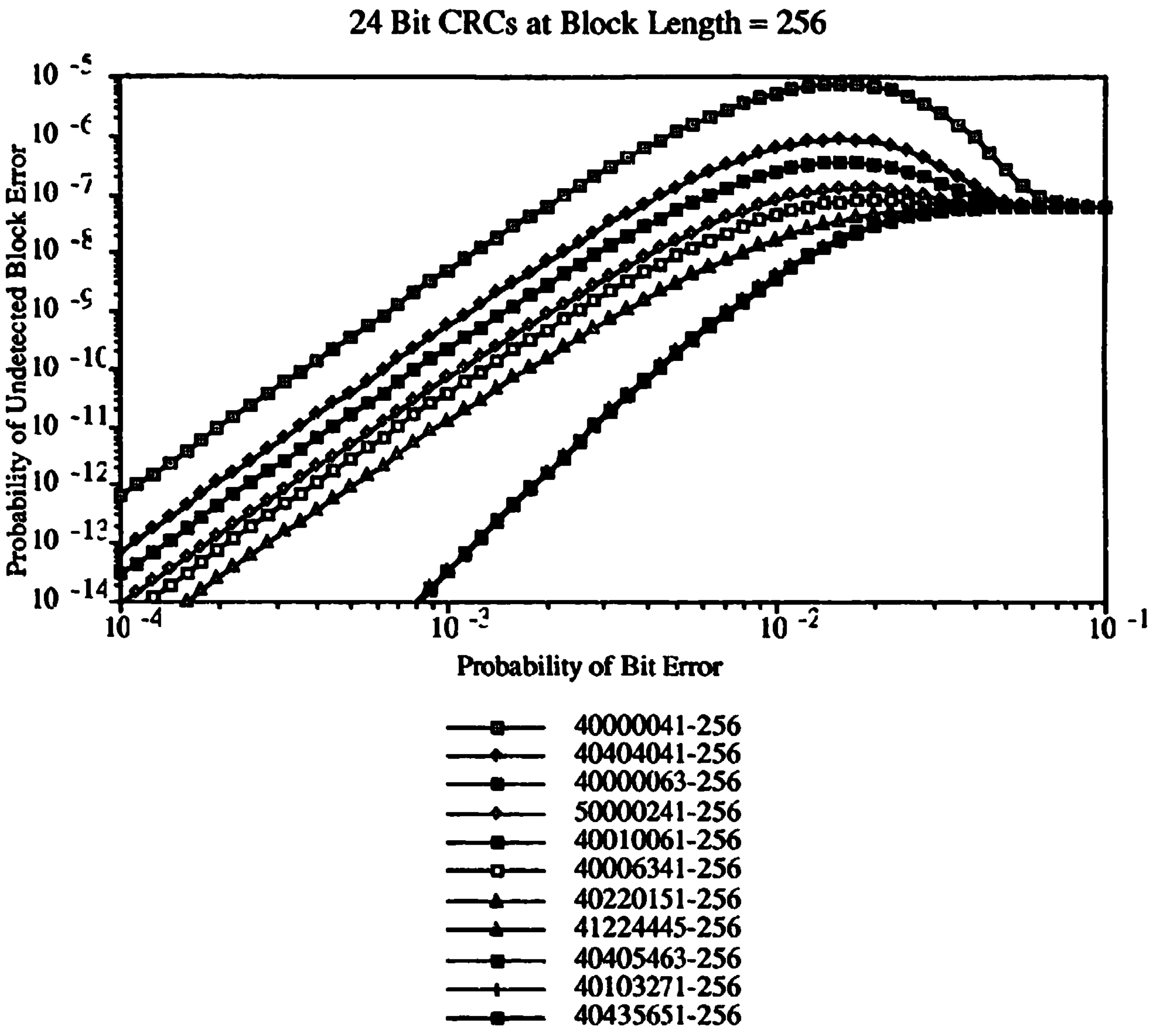


Figure 12. Probability of undetected block error vs. probability of bit error for 24 bit CRCs at block length n = 256.

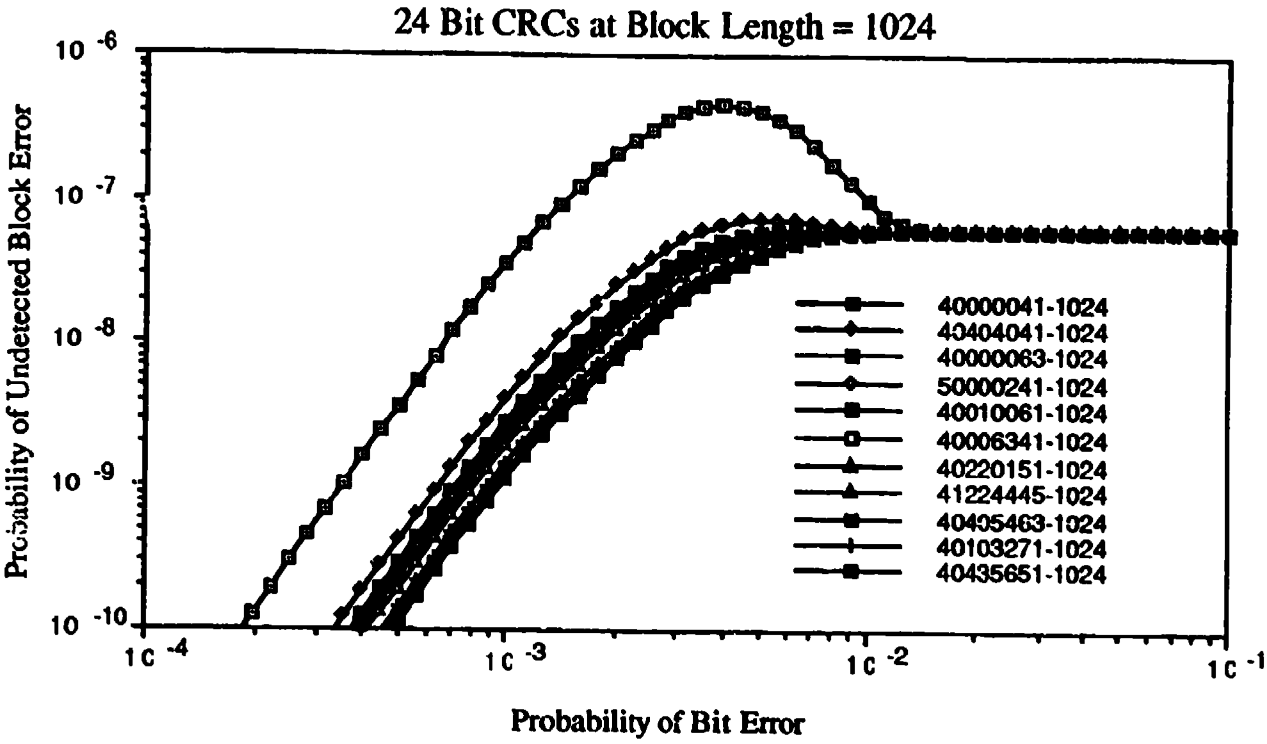


Figure 13. Probability of undetected block error vs. probability of bit error for 24 bit CRCs at block length n = 1024.

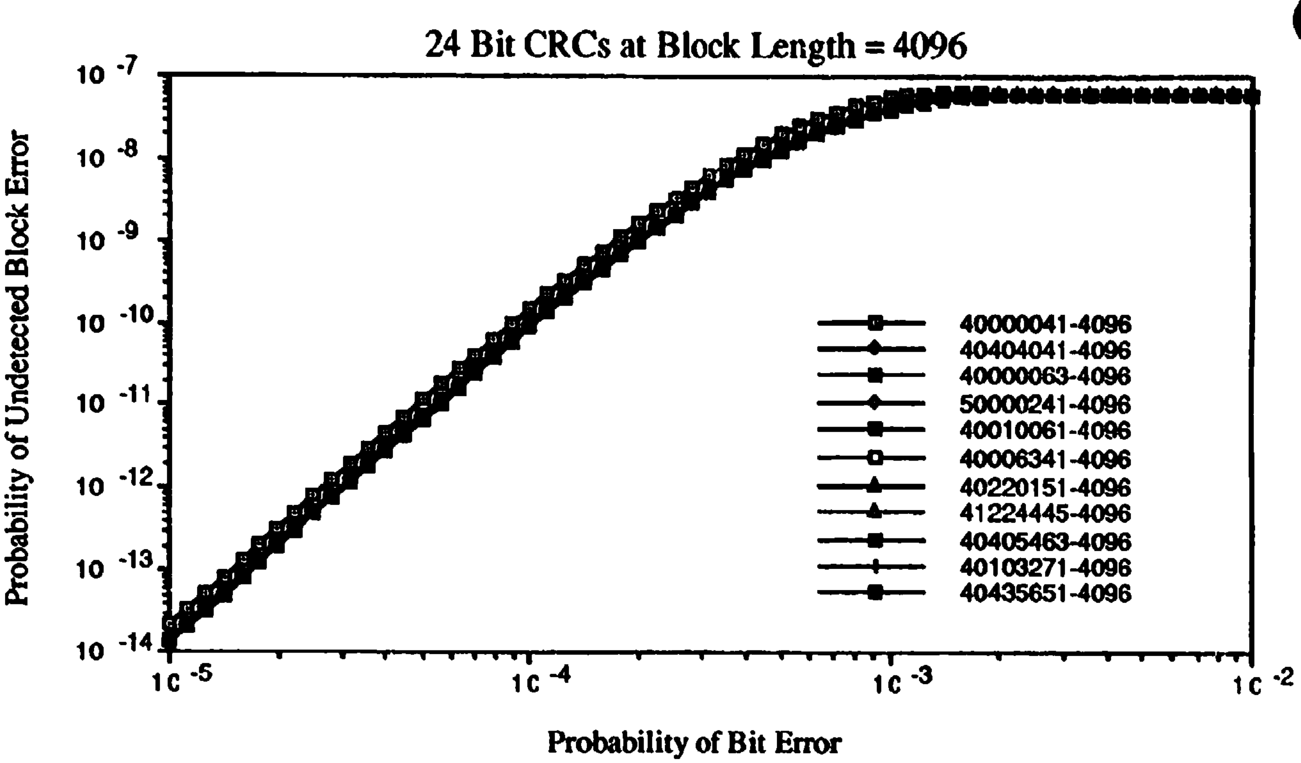


Figure 14. Probability of undetected block error vs. probability of bit error for 24 bit CRCs at block length n = 4096.

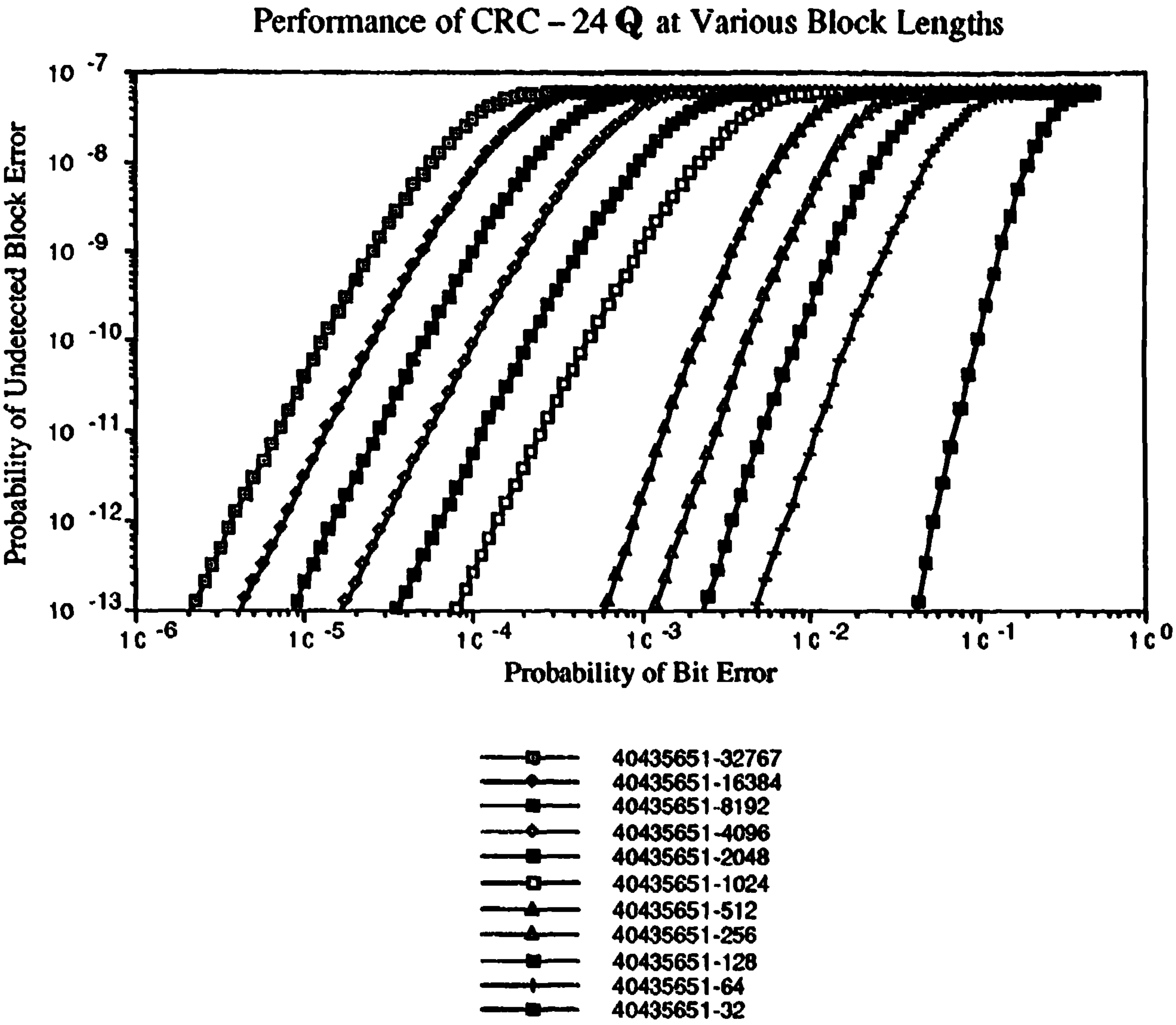


Figure 15. Probability of undetected block error vs. probability of bit error for CRC-24Q at various block lengths.

X. EVALUATION OF 32 BIT CRCS

As in section IX, we tested 32 bit CRCs generated from the degree 31 primitive irreducible polynomials listed in Peterson and Weldon [5]. The list contains a total of 11 polynomials. However, because of time limitations we were not able to test these 32 bit CRCs extensively at a large number of block lengths. At each block length that we did test at, the probability of undetected error was evaluated for twenty logarithmically spaced values in each decade for p, from $p = 10^{-6}$ to $p = 1/2$.

We were able to reject all but one of the polynomials as being significantly improper at some block length tested. The lone survivor is the 32 bit CRC, CRC-32Q generated from the primitive polynomial 20060140231.

In Figure 16, we plot the error detection performance of 32 bit CRCs generated from the 11 polynomials considered at block length 1024. In Figures 17, 18, we plot the performance of 32 bit CRCs generated from the best three of these polynomials, which is indistinguishable in Figure 16, at block lengths 256 and 4096, respectively.

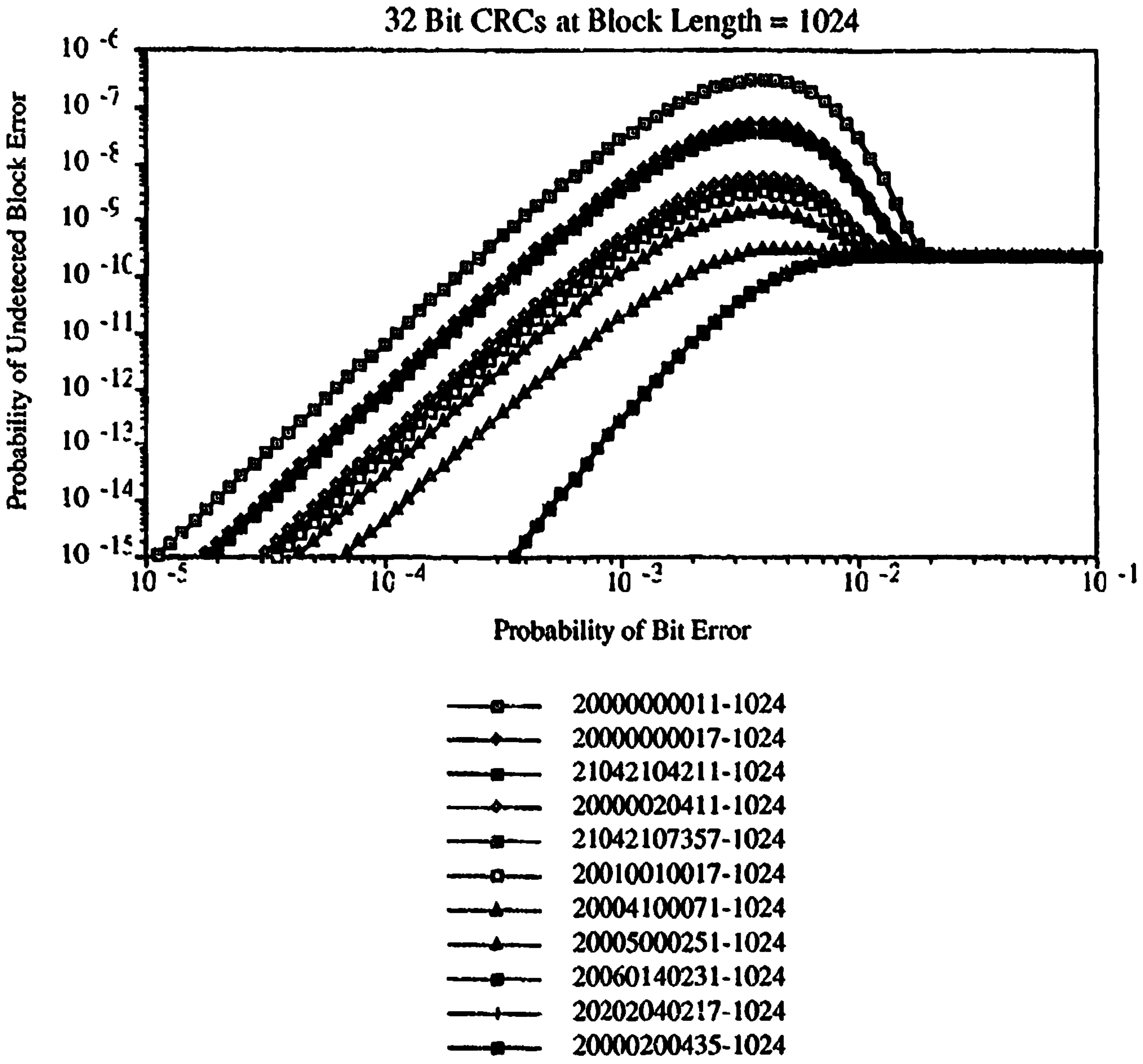


Figure 16. Probability of undetected block error vs. probability of bit error for 32 bit CRCs at block length n = 1024.

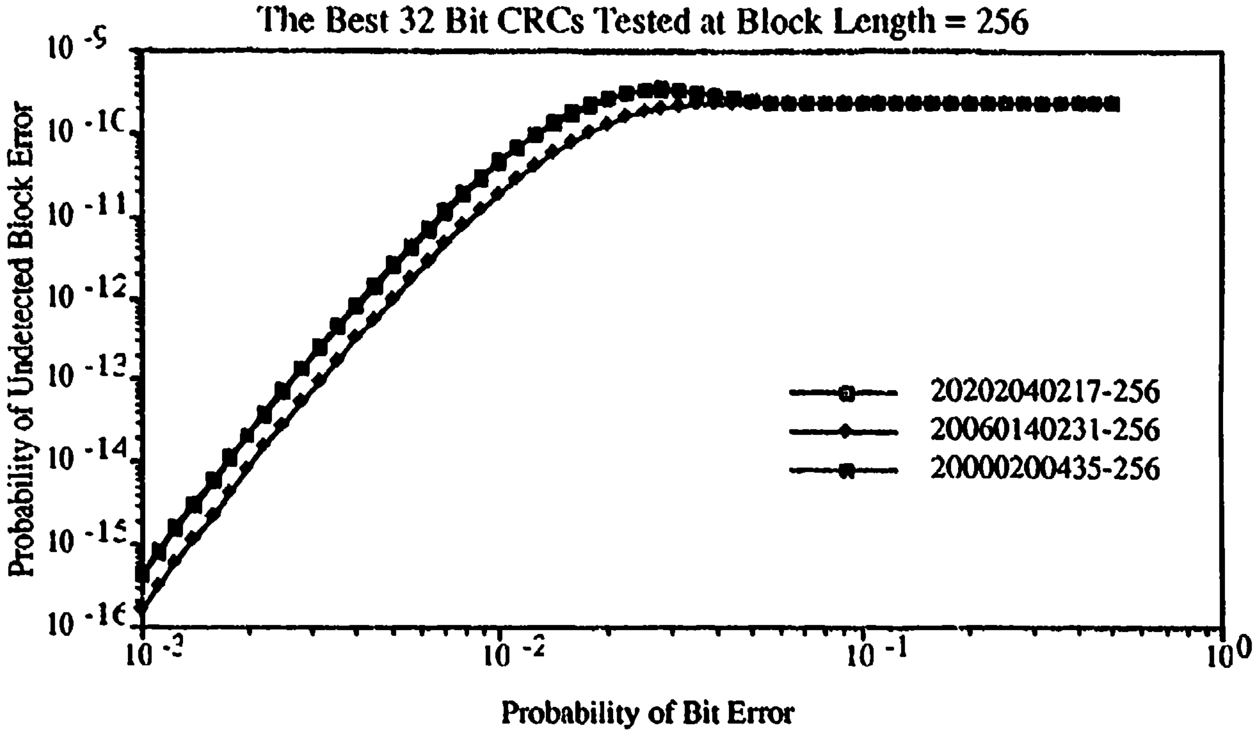


Figure 17. Probability of undetected block error vs. probability of bit error for 32 bit CRCs at block length n = 256.

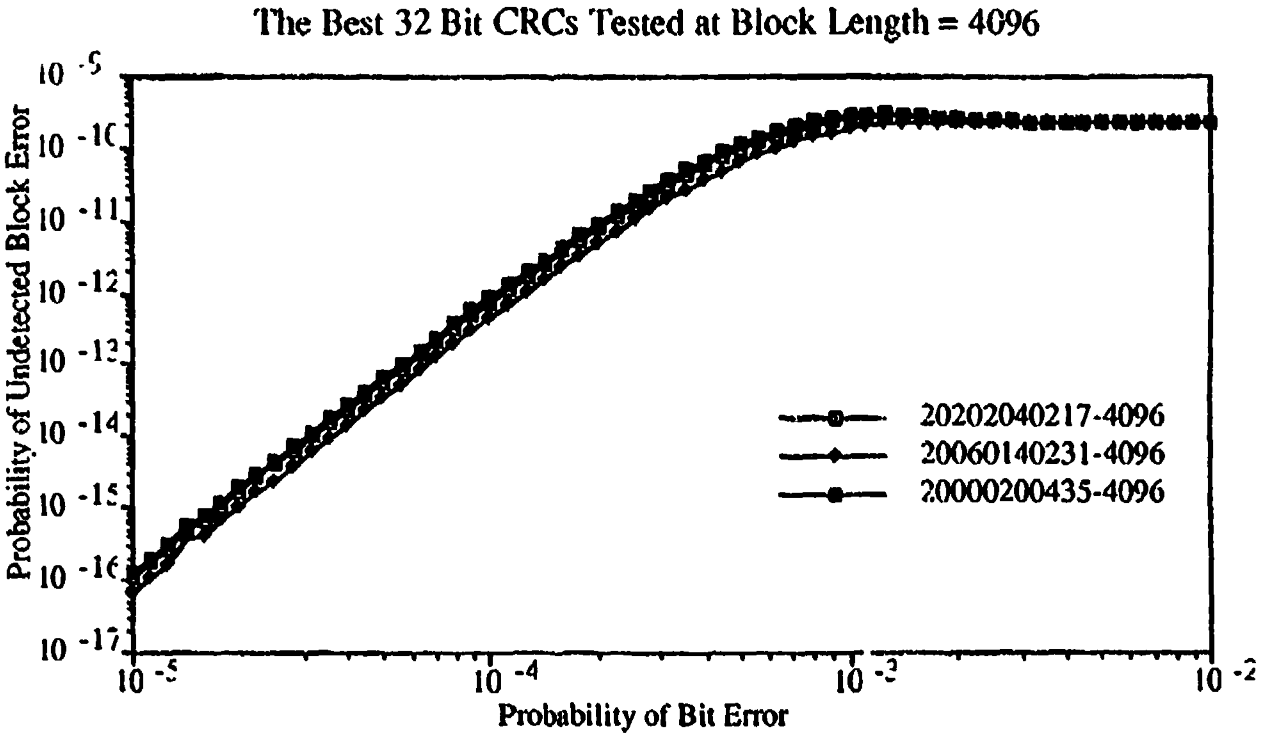


Figure 18. Probability of undetected block error vs. probability of bit error for 32 bit CRCs at block length n = 4096.

XI. CONCLUSION

In this report, we summarize the results of a Phase I study of the error detection capability of shortened CRC codes of the form (x + 1)p(x), where p(x) is a primitive irreducible polynomial. The evaluation algorithm used and its optimization in software is explained in detail.

We conclude that, for moderate redundancy CRCs, it is reasonable to distinguish those CRCs which will guarantee a certain degree of error detection capability in the presence of random bit errors at all shortened block lengths from those CRCs which exhibit anomalies in their error detection performance at shortened block lengths using our algorithm. In section VIII, we give an exhaustive list of 16 bit CRCs which are proper at all shortened block lengths.

As the number of redundant bits increases, it becomes more difficult to completely evaluate all choices of primitive polynomials at all block lengths. We have demonstrated, however, that it is feasible to examine specific polynomials, eliminate improper CRCs, and find CRCs which perform well at a wide variety of block lengths. Specifically, in section IX we recommend as a potential new standard a 24 bit CRC, CRC-24Q, which has been extensively tested.

We also report on some unfavorable results which seem to indicate that it is not possible to make accurate projection of the error detection performance of shortened CRC codes based on a small sample of their code words.

Finally, we give an informal proposal for an improved shortened CRC code evaluator, which would enable us to more thoroughly examine CRCs with a large number of redundant bits, based on building specialized hardware.

XII. kEFERENCES

[1] Tanenbaum, A.S., Computer Networks, Prentice-Hall, Inc., Englewood Cliffs, W.J., 1981.

[2] Blahut, Richard E., Theory and Practice of Error Control Codes, Addison-Wesley Publishing Company, Inc., May 1984.

[3] Wolf, J.K., Michelson, A., and Levesque, A., "On the Probability of Undetected Error for Linear Block Codes," IEEE Trans on Communications, vol. COM-30, pp. 317-324, Feb. 1982.

[4] Chang, S.C. and Wolf, J.K., "A Simple Derivation of the MacWilliams Identity for Linear Codes," IEEE Trans on Information Theory, vol. IT-26, pp. 476-477, July 1980.

[5] Peterson, W. Wesley and Weldon, E.J., Jr., Error-Correcting Codes, Second Edition,The MIT Press, Massachusetts Institute of Technology, October 1980.

[6] Golomb, Solomon W., Shift Register Sequences, Holden-Day, Inc., 1967.

[7] Elspas, Bernard, The Theory of Autonomous Linear Sequential Networks, IRE Transactions of the Professional Group on Circuit Theory, vol. CT-6, Number 1, March 1959.

[8] Aho, Alfred V., Hopcroft, John E., and Ullman, Jeffrey D., The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Company, 1974.

XIII. PROFESSIONAL PERSONNEL

The principal investigators for this study were Dr. Andrew J. Viterbi, Dr. Jack K. Wolf, Senior Engineering Associate and Mr. Lyle J. Fredrickson, Engineer. Dr. Viterbi is a co-founder and Chief Technical Officer of QUALCOMM, Inc. Dr. Wolf is an exclusive consultant to QUALCOMM, Inc. and devoted one day a week to these consulting efforts.

ANDREW J. VITERBI
VICE CHAIRMAN AND CHIEF TECHNICAL OFFICER

Education:

Andrew J. Viterbi received the S.B. and S.M. degrees in Electrical Engineering from the Massachusetts Institute of Technology in 1957. He received the Ph.D. degree in Electrical Engineering from the University of Southern California in 1962.

Experience:

Dr. Viterbi has devoted approximately equal segments of his career to academic research, industrial development, and entrepreneurial activities.

In his first employment after graduating from MIT in 1957, he was a member of the project team at C.I.T. Jet Propulsion Laboratory which designed and implemented the telemetry equipment on the first successful U.S. satellite, Explorer I. In the early sixties at the same laboratory, he was one of the first communication engineers to recognize the potential and propose digital transmission techniques for space and satellite telecommunication systems.

As a professor in the UCLA School of Engineering and Applied Science from 1963 to 1973, he did fundamental work in digital communication theory and wrote two books on the subject, for which he received numerous professional society awards and international recognition. These include three paper awards, culminating in the 1968 IEEE Information Theory Group Outstanding Paper Award. He has also received three major society awards: the 1975 Christopher Columbus International Award (from the Italian National Research Council sponsored by the City of Genoa); the 1980 Aerospace Communications Award jointly with Dr. Irwin Jacobs (from AIAA); and the 1984 Alexander Graham Bell Medal (from IEEE sponsored by AT&T) "for exceptional contributions to the advancement of telecommunications".

The practical development of these theoretical principles led to the founding of LINKABIT Corporation, together with Dr. Irwin Jacobs. Between 1968 and 1980, LINKABIT grew to become a sizable company with unique expertise and leadership in digital signal processing equipment for military and government satellite communication. The company has grown at the same rate since being acquired by M/A-COM Inc. in 1980, expanding into commercial telecommunication equipment, terrestrial as well as satellite, and video scrambling business areas. Dr. Viterbi was Executive Vice President of LINKABIT from 1974 to 1982. In 1982, when Dr. Jacobs became Executive Vice President of M/A-COM, he took over as President of M/A-COM LINKABIT, Inc. From 1984 to 1985, he was appointed Chief Scientist and Senior Vice President of M/A-COM, Inc.

On July 1, 1985, Dr. Viterbi became founder a.id Vice Chairman and Chief Technical Officer of QUALCOMM, Inc.

Dr. Viterbi is a member of the U.S. National Academy of Engineering and a Fellow of IEEE. He is past Chairman of the Visiting Committee for the Electrical Engineering Department of Technion, Israel Institute of Technology, and he is presently a member of the MIT Corporation Visiting Committee for Electrical Engineering and Computer Science. He is also past Chairman of the U.S. Commission on Signal Processing of the International Radio Scientific Union (URSI) and a past member of the Army Science Board. He has long been active in IEEE, having served two terms of the Board of Governors of the Information Theory Group, chairing the Board in 1970 and he served as its Transactions Associate Editor for a term. He has been invited as a Distinguished Lecturer by the University of Illinois Coordinated Sciences Laboratory and he has been honored as the 1986 Outstanding Engineering Graduate by the University of Southern California School of Engineering. In spite of his corporate administrative duties, he has managed to remain technically current, having recently proposed new spread spectrum processing techniques for jam resistant communications and for digital cellular radio.

Professional Credits:

1-33. List available upon request.

34. "Spread Spectrum Communications - Myths and Realities," *IEEE Communications Magazine*, pp. 11-18, May, 1979.

35. "Interleaving and Coding for Satellite Channels Perturbed by Pulsed RFI," (with I. Bar-David and J.P. Odenwalder), *ICC '80 Conference Record*, Volume 1, pp. 4.2.1-4.2.6, Seattle, Washington, June 16-18, 1980.

36. "Multiple Access Communication Using Coded Pulse Interval Modulation," (with I. Gurantz, S. Gardner and E. Zbik), *NTC '80 Proceedings*, Houston, Texas, pp. 14.4.1-14.4.5, December, 1980.

37. "Coding and Interleaving for Correcting Burst and Random Errors in Recording Media", Proceedings of Digital Audio Conference, Rye, N.Y., June 3-6, 1982.

38. "A Robust Ratio-Threshold Technique to Mitigate Tone and Partial Band Jamming in Coded MFSK Systems," *Proceedings of IEEE Conference on Military Communication*, MILCOM '82, Boston, Massachusetts, October 17-20, 1982.

39. "Nonlinear Estimation of PSK-Modulated Carrier Phase with Application to Burst Digital Transmission," (with Audrey M. Viterbi), *IEEE Transactions on Information Theory*, Vol. IT-29, No. 4, July 1983.

40. "Robust Decoding of Jammed MFSK/FH Modulation" (with T. Schonhoff and M. Mulligan), *IEEE Conference of Military Communication*, MILCOM '84, Los Angeles, California, September 1984.

41. "When Not to Spread Spectrum - A Sequel" *IEEE Communications Magazine*, Vol 23, No. 4, pp 12-17, April 1985.

42. "Robust Decoding of Jammed Frequency Hopped Modulation" in *Recent Advances in Communication and Control Theory*, Edited by R.E. Kalman, et. al., Optimization Software, Inc., New York, 1987.

Books:

1. *Digital Communications with Space Applications*, (co-author with S.W. Golomb, L.D. Baumert, M.F. Easterling, and J.J. Stiffler), Prentice-Hall, 1964.
2. *Principles of Coherent Communication*, McGraw-Hill, 1966.
3. *Advances in Communication Systems*, Vol. 4 (editor), Academic Press, 1975.
4. *Principles of Digital Communication and Coding*, (with J.K. Omura), McGraw-Hill, 1979.

JACK K. WOLF
ENGINEERING ASSOCIATE

Education:

Jack Keil Wolf is a chaired professor in the Center for Magnetic Recording Research at the University of California, San Diego, La Jolla, California. He received his B.S.E.E. degree from the University of Pennsylvania in 1956, and his M.S.E., M.A., and Ph.D. degrees from Princeton in 1957, 1958, and 1960, respectively.

Experience:

Dr. Wolf was a member of the EE Department at New York University from 1963 to 1965, and the Polytechnic Institute of Brooklyn from 1965 to 1973. He was chairman of the Department of Electrical and Computer Engineering at the University of Massachusetts, from 1973 to 1975 and was Professor there from 1973 to 1984. Since 1985 he has been a Professor of Electrical and Computer Engineering and a member of the Center for Magnetic Recording Research at the University of California, San Diego. During the 1968-1969 academic year, he was a member of the Mathematics Research Center at Bell Laboratories. From 1971 to 1972 he was an NSF Senior Postdoctoral Fellow at the University of Hawaii. From 1979 to 1980 he was a Guggenheim Fellow at the University of California at San Diego and the LINKABIT Corporation. His research interests are in information theory, coding theory, communications systems, computer networks and magnetic recording.

Dr. Wolf is a fellow of the IEEE. He was co recipient of the 1975 IEEE Information Theory Group Paper Award for the paper "Noiseless Coding of Correlated Information Sources" (coauthored with D. Slepian). He was cochairman of the 1969 IEEE International Symposium on Information Theory. He served on the Board of Governors of the IEEE Information Theory Group from 1970 to 1976 and from 1980 to 1986. Dr. Wolf was president of the IEEE Information Theory Group in 1974. He was International Chairman of Commission C of URSI from 1980 to 1983.

Dr. Wolf began a part-time association with QUALCOMM, Inc. in October 1986. His expertise in coding and networks has been applied to a variety of programs. These include the synthesis and subsequent analysis of the ARNS (Adaptive Receive Node Scheduling) protocol for the MSS program, the development of a new algorithm for optimally separating the tracks of multiple targets in a radar system with very high false alarm rate, and the development of trellis coded modulation systems for a variety of applications including Class IV partial response channels.

Publications:

1-111 List available on request.

112. "Combined Error Correction/Modulation Codes", (with P. Lee), *1987 Intermag Conference Digest* (to be published in *ıEEE Trans. on Magnetics*).

113. "On the Performance Evaluations of Trellis Codes" (with E. Zehavi), *IEEE Trans on Information Theory*, Vol. IT-33, March 1987.

114. "Multi-Group Random Access System" (with M. Rodriquez and R. Rao), *IEEE INFOCOM '87 Conference Record*, April 1987.

115. "A Robust Collision Resolution Algorithm for the Random Access System with a Noisy Channel" (with K. Ho and R. Rao), *IEEE INFOCOM '87 Conference Record*, April 1987.

116. "On Runlength Codes" (with E. Zehavi). Accepted for publication in *IEEE Trans. on Information Theory*, 1988.

117. "A Class of Binary Burst Error-Correcting Quasi-Cyclic Codes" (with W. Zhang). Accepted for publication in *IEEE Trans. on Information Theory*, 1988.

118. "On Saving Decoder States for Some Trellis Codes and Partial Response Channels" (with E. Zehavi). Accepted for publication in *IEEE Trans. on Communications*, February, 1988.

119. "Signalling with Special Run-Length Constraints for a Digital Recording Channel" (with C. French and G. Dixon), Accepted for publication in *IEEE Trans on Magnetics*, 1988.

120. "Collision Resolution Algorithms for a Time-Constrained Multi-Access Channel" (with S. Panivar, P. Towsely and A. Armoni), *Proceedings of the Twenty-Fifth Annual Conference on Communication, Control and Computing*, September 1987.

121. "Trellis Decoding and Applications to Multi-Target Tracking" (with A.M. Viterbi and G. Dixon), *SPIE's Symposium on Innovative Science and Technology*, Los Angeles, CA, January 1988.

LYLE J. FREDRICKSON
ENGINEER

Education:

Lyle J. Fredrickson received the MSEE degree in Electrical and Computer Engineering from the University of California, San Diego in 1988. He received the BA degree in Mathematics from the University of California, Berkeley in 1980. He will receive the Ph.D. degree in June 1989.

Experience:

Mr. Fredrickson was an Associate Engineer at Cyclotomics, Inc. from January 1985 to August 1986. He supervised all aspects of production of stock units and development systems for digital forward error correction, using Reed-Solomon codes. He designed and produced CMOS versions of model 888 encoders and decoder.

Prior to that, from August 1984 to December 1984, Mr. Fredrickson worked at Codex Corporation on telephone interfaces using digital signal processors.

Papers Presented:

"Coding Using Multiple Block (d,k) Codes," International Conference on Communications (ICC '89), Boston, MA, June 1989.

"Error Detecting Multiple Block (d,k) Codes," Intermag '89, Washington, DC, March 1989.